بسم الله االرحمن الرحيم



الجزء الاول.

كتابة: خالد ناصيف داود عبدالقادر.
طالب سنة ثانية في جامعة البلقاء التطبيقية .
تخصص هندسة برمجيات.
khalid.nassif_se@yahoo.com

•

اللهم لا علم لنا الا ما علمتنا.
بسم الله والصلاة والسلام على رسول الله سيدنا محمد والحمد لله رب العالمين الذي جعلني سببا في كتابة هذا الكتاب اما بعد فهذا الكتاب هو عبارة عن بعض الافكار في هياكل البيانات وشرحها واتمنى ان اكون قد اوفيت بالشرح ولم اقصر في شيء وفكرة هذه الكتاب اتت لانه لايوجد اي كتاب عربي عن هياكل البيانات في السي++ فاتت الضرورة لوجود كتاب عربي عن هذه الافكار في السي++ وطبعا وللاسف هذا الكتاب لا يوجد به الشرح الكامل لهياكل البيانات وباذن الله سوف يكون هناك جزء اخر يكون تكملة لهذا الكتاب والذي نقص في هذا الكتاب هو الاشجار (trees) وتحليل البرامج بطريقة ال (big-o) وقد ركزت اكثر شيء على الكودات واهم شيء في هياكل البيانات ان تكون على قدر جيد من فهم الكود وان تكون قادر على كتابة الكود لوحدك .
مع العلم بان هياكل البيانات هي ليست شيء جديد في السي++ ولكنها افكار ومهارات وطرق في استخدام لغة السي++ و اي شيء تريد عمله في هياكل البيانات المهم ان تفهم مبدأ الفكرة.

ارجو ان استقبل اراءكم الصريحة على الايميل واكون شاكرا لاي نقد بحق هذا الكتاب وايضا لاي استفسارات او اي امثلة تصعب عليكم او اي سؤال عام على ال ++c بخصوص الكتاب او خارجه راسلني على الايميل وباذن الله اذا كنت اعرف الاجابة فاني لن اقصر khalid.nassif_se@yahoo.com

•

وسوف نتناول في هذا الكتاب المواضيع التالية:

•

ما المطلوب منك كي تستطيع قراءة هذا الكتاب؟
كل ما يجب عليك معرفته هو كيفية البرمجة في ال object oriented وايضا تكون على دراية كافية باساسيات ال c++ وغير ذلك اي شيء يكون لديك فيه ضعف وخاصة المؤشرات لانني لم اقم بشرحها الشرح الكافي فانني انصح بالمراجعة من كتاب الاكسير لانه كتاب يعتبر افضل الكتب العربية بال c++ وباذن الله اكون قد شرحت بطريقة سلسة ومفهومة .

ماذا استفيد من هياكل البيانات ؟
هياكل اليانات صحيح انها فقط عبارة عن افكار ولكنك ايضا تدخل في تفاصيل لغة ال c++ فتصبح تعرف كيف تعمل البرامج بطريقة بسيطة وفعالة وطرق لحل المشاكل وغيرها من اشياء مفيدة وطبعا البرمجة تعتمد على اساس حل المشكلات وهياكل البيانات هي الصميم في حل المشكلات .

ماذا سوف يكون بالجزء الثاني من الكتاب؟
سو ف يكون شرح بقية هياكل البيانات وايضا الدخول في مكتبات ال c++ اي ان ندخل في مكتبات ال c++ نفهم كيف تجري العمليات في هذه اللغة الفعالة وباذن الله سوف يتم نشره في ما بين 17/9/2008 و 01/10/2008 وسوف يكون بنفس الاسم.

لماذا هياكل البيانات بال c++ ؟
ممكن ان يكون راي متطرف وذلك من محبتي لهذه اللغة ولكن لغة ال c++ باعتقادي هي افضل لغة واذا لم تكن افضل لغة فهي افضلها في هيكلة البيانات وذلك لاحتوائها على المؤشرات وهذا ما لا تحتويه العديد من اللغات الاخرى مثل ال java ففي ال c++ نستطيع ان نتحكم بالذاكرة تحكم كالم وبسيط بنفس الوقت فباعتقادي هذا هو افضل شيء على الاطلاق.

•

(بعض المهارات في السي++)

1-المصفوفات عن طريق استخدام vector :

وهي مصفوفة عادية فيها عناصر وتبدا من 0 حتى حجم المصفوفة ناقص 1 اي انها مثل المصفوفات العادية ولكن هنا في ال vector انها مصفوفة اكثر تقدما ومرنة وايضا ويوجد لها دوال ومن اهم الدوال انه يوجد دالة resize وهي انه في اي وقت من الاوقات تستطيع ان تغير حجم المصفوفة حتى خلال تنفيذ البرنامج من الممكن ان تجعل المستخدم يدخل عدد العناصر الذي يريدها وهذا شيء غير موجود في المصفوفات الكلاسيكية وطبعا هذه الخاصية مفيدة جدا.

ومن اجل استخدام ال vector يجب ان نضمن مكتبة <vector> وبما انها ليس h. فاننا يجب ان نضع جملة using namespace std; وبعد ذلك عند تعريف مصفوفة فاننا اول شيء نضع vector اي انها من نوع vector ومن ثم نضع نوع البيانات بين قوسين منحرفين فاذا كنا نريد int فاننا نضعها هكذا <int> وبعد ذلك نضع اسم المصفوفة ومن ثم نضع حجم المصفوفة كبين قوسين عاديين .

```
vector <double> back (20);
```

فهنا عرفنا مصفوفة من نوع vector نوع بياناتها double واسمها back وحجمها 20 .

***** عند تعريف مصفوفة من نوع vector فانها تعطي جميع عناصر المصفوفة قيمة 0 كقيمة ابتدائية.**

لنأخذ مثال :---

•

```
#include <iostream.h>
#include <vector>
using namespace std;
void main()
{
int a;
vector <int> password(10);
password[1]=5;
cout<<"the second element::
"<<password[1]<<endl;
cout<<"the first element ::
"<<password[0]<<endl<<endl<<endl;
vector <int> mam(10,7);
mam[1]=5;
cout<<"the second element:
"<<mam[1]<<endl;
cout<<"the first element ::  "<<mam[0]<<endl;
cout<<"the size of the mam vector ::
"<<mam.size()<<endl;
cout<<"enter number toresize the mam vector
::..::"<<endl;
cin>>a;
mam.resize(a);
cout<<"the size of mam vector now ::
"<<mam.size()<<endl;
}
```

قمنا بتعريف مصفوفة vector واسميناها password ونوع بيانتها int وحجمها 10 وبعد ذلك قلنا له ان اجعل العنصر الذي رقمه واحد يساوي 5 وطبعا بما ان المصفوفة تبدا من الصفر فان العنصر رقم واحد هو العنصر الثاني .

•

وبعد ذلك قلنا له ان يطبع العنصر رقم 1 ثم قلنا له ان يطبع العنصر رقم 0 وهو العنصر الاول وهنا في العنصر الاول طبع لنا صفر وذلك لاننا قلنا ان المصفوفة اول ما نعرفها انها تضع قيمة 0 في كل عناصرها .

وبعد ذلك عرفنا مصفوفة vector نوعها int وحجمها 10 ولكن هناك شيء جديد وهو انه بعدما وضعنا الحجم فاننا وضعنا رقم 7 بعد فاصلة وهو كاننا نقول له ان حجمها 10 واملئ عناصر المصفوفة كلها بالقيمة 7 فهنا بدل ان يضع القيمة الابتدائية للمصفوفة 0 فان القيمة الابتدائية اصبحت 7 .

ومن ثم قلنا له ان العنصر رقم 1 ضع به القيمة 13 .

ثم طبعنا القيمة التي بالعنصر 1 ومن ثم طبعنا القيمة التي في العنصر رقم 0 وطبعا سوف يكون في العنصر رقم 0 القيمة 7 لاننا جعلنا القيمة الابتدائية للمصفوفة هي 7.

ثم هنا في جملة mam.size() فان size هو دالة في ال vector وهو يرجع لنا حجم المصفوفة .

وبعد ذلك ادخل المستخدم قيمة في المتغير a .

ولاحظ جملة mam.resize(a) فانها جملة تغيير حجم المصفوفة و resize هي دالة لل vector وهنا بين القوسين نضع حجم المصفوفة وكما راينا اننا نستيع جعل حجم مصفوفة بتحكم من المستخدم وهذا ما لا تستطيعه المصفوفات الكلاسيكية .

وهناك دوال اخرى جميلة في ال vector مثل اعطاء اول قيمة واخر قيمة وشطب المصفوفة وما الى ذلك من دوال مفيدة وتستطيع معرفة هذه الدوال انه بعد ان تكتب اسم المصفوفة وتضع نقطة فانها سوف تظهر لك في قائمة .

*** اذا لم ينجح البرنامج في التنفيذ معك وقال لك ان هناك خطا فحاول ان تعمل تضمين للمكتبة <apvector.h> بدل <vector> والغي جملة using namespace std; وعند تعريف مصفوفة ضع كلمة apvector بدل vector .

****

•

2- جملة تعريف انواع البيانات (typedef) :

جملة typedef هي جملة من اجل ان نعرف نوع بيانات معين (اي int او char الخ..) باسم اخر .

وتكون صيغة الجملة اول شيء كلمة typedef ثم نوع البيانات ثم الاسم الذي تريد استخدامه.

مثال:----

```
#include <iostream.h>
typedef int entry;
void main()
{
typedef int entry;
entry y;
entry m;
entry s;
cin>>y>>m>>s;
cout<<y<<endl<<m<<endl<<s<<endl;
}
```

فهنا قد عرفنا نوع البيانات int بالاسم entry .اي انه اينما توضع كلمة entry فان البرنامج ياخذها على انها int .

وكما نلاحظ اننا قمنا بتعريف المتغيرات بال entry .

والفائدة منها هو انه اذا اردنا تغيير نوع البيانات فاننا فقط نغير النوع الذ ي عرفنا به الاسم فبالمثال السابق اذا اردنا تحويل البيانات من int الى char كل ماعلينا فعله هو ناتي للجملة التي عرفنا بها الاسم entry ونغير النوع الى char اذن في المثال السابق تصبح جملة ال typedef هكذا:

```
typedef char entry;
```

فتخيل انه انك كتبت كود من الف سطر وتريد تغيير النوع البيانات التي تريد ادخالها فهل سوف تذهب الى كل تعريف وتغيره طبعا لا والى اصبحت لغة السي ++ باليه فالذي تفعله انك تعرف اسم من

•

نوع ما وتستخدمه عند تعريف المتغيرات التي من الممكن ان تقوم بتغيير نوع بياناته.
****

3- تعدد الاحتمالات و جملة (enum) :
يوجد مواقف كثيرة تكون احتمالاتها الذي سوف تنتهي به اكثر من حالتين كنا عند الحالتين نستخدم ال bool ولكن اذا كان هناك اكثر من حالتين فماذا سنفعل وجاء الحل في جملة enum وهذه الجملة تعطيك صلاحية تعريف الحالات التي تريد و عدد الحلات التي تريد.
فيصبح باستطاعتك تعريف متغيرات من نوع ال enum الذي عرفته.
وصيغتها كالتالي اول شيء كلمة enum ثم نفتح قوس } ثم نضع الحالة الاولى ثم فاصلة , ثم الحاله الثانية ثم فاصلة , ...وهكذا حتى انتهاء الحالات نضع تسكيرة القوس { وفي النهاية نضع فاصلة منقوطة ; .
مثال :---

```
#include <iostream.h>
enum error{success,notgood,theworst};
error check(int);
void main()
{
int a;
cin>>a;
error r=check(a);
if (r==success)
{
cout<<"success"<<endl;
}
else if(r==notgood)
```

```
{
cout<<"notgood"<<endl;
}
else
cout<<"the worst"<<endl;
}
error check (int s)
{
if (s>90)
return success;
else if (s<90 && s>60)
return notgood;
else
return theworst;
}
```

وفي المثال السابق قمنا بتعريف error من نوع enum وفيها ثلاث حالات الحالة الاولى success والثانية notgood والثالثة theworst .

**** تستطيع استخدام الانواع التي تعرف بها ال enum ان تعرف بها المتغيرات وتعف بها الدوال ايضا.****

****

4- عمليات التاكد من شروط البرنامج وجملة (assert):

assert هي عبارة عن اداة (debug) وهي اداه للتاكد من الاخطاء وتصحيح البرامج . عند استخدامها يجب تضمين مكتبة assert.h او مكتبة cassert احداهما فقط ليس الاثنتين.
فهنا جملة assert هي عبارة عن جملة شرطية ترجع حالتان true او false فاذا رجعت true فانه يكمل البرنامج اما اذا ارجع false فان البرنامج يتوقف ويخرج لنا جملة error ويخرج لنا

•

شاشة سوداء يقول لنا فيها انه هناك خطا في الشرط كذا في السطر كذا.

مثال:---

```
#include <iostream.h>
#include <assert.h>
void main()
{
int a;
cin>>a;
assert(a<5);
cout<<"we come over it::..::"<<endl;
}
```

ففي المثال السابق اذا ادخلنا له عدد اقل من 5 فان البرنـامج يكمل ما له اما اذا ادخلنا رقم اكبر من 5 فان البرنـامج يتوقف ويخرج error ثم يخرج شاشة سوداء فيها الاتي :

assertion failed: a<5, file وهنا يعطيك موقع الملف في جهازك , line 7

اذن اعطى انه هناك خطأ في احدى جمل ال assert وذلك عند شرط a<5 في الملف الذي موقعه كذا عند السطر ال 7 .

**** يوجد هنا فائدة جيدة هي انه يعطينا الملف الذي حصل فيه الخطأ فانه اذا كان البرنـامج يتكون من اكثر من ملف فانه يعطيك باي ملف حصل الخطا .**

اذن فهي تستخدم عندما نريد ان نتحقق من ان شيء معين صحيح ام لا او ان شيء ما لم يخرج عن نطاقه فنبقي البرنـامج متماسك بهذه الطريقة ونجعله بعيد عن الخطأ قدر الامكان.

****

5- ملاحظة على ال objects :

```
#include <iostream.h>
class a
{
public:
```

•

```
a(int m)
{
_m=m;
cout<<"iam the a "<<_m<<" constructer
::..::"<<endl;
}
~a()
{
cout<<"iam the a "<<_m<<" destructer
::..::"<<endl;
}
private:
int _m;
};
a a1(1);
void main()
{
a a2(2);
}
```

هنا قمنا بتعريف object قبل ال main (اي في المنطقة العامة للبرنامج) وعرفنا object داخل ال main فان البرنامج سوف يبدأ بتنفيذ ال constructer لل object الذي خارج ال main ثم يبدا بالذي داخل ال main ثم عند النهاية فانه ينفذ ال destructer للذي داخل ال main ثم الذي خارج ال main.
ففي المثال السابق سوف ينفذ constructer ال a1 الذي هو في المنطقة العامة ومن ثم ينفذ conatructer ال a2 الذي داخل ال main وعند النهاية سوف ينفذ destructer ال a2 ومن ثم ينفذ destructer ال a1 .
فسوف يكون اخراج البرنامج لنا الاتي:

•

```
iam the a 1 constructer ::..::
iam the a 2 constructer ::..::
iam the a 2 destructer ::..::
iam the a 1 destructer ::..::
```

***بعضكم سوف يقول انه خرج لي الاتي ولكن اخر جملة لم تخرج التي هي(iam the a 1 destructer ::..::)
وذلك بانه يكون برنامج الترجمة الذي تستخدمه (compiler) لا يعتمد السي++ الاساس . والاغلب انك تستخدم برنامج vc++ v.6.0 فان به هذا الخطا ايضا.

6- التحويل من int الى char او string والعكس :
في بعض ا لاحيان هنالك تكون الارقام على موجودة لديك على شكل احرف اي انها داخل متغير من نوع char وتريد تحويلها الى صيغة ال int وهنا ابسط حل لهذه المشكلة هو اننا نقوم بطرح المتغير من '0' كالآتي :

```
char  id='9';
int _idno=id-'0';
```

اما اذل اردت ان تفعل العكس اي ان تحول الارقام وتجعلها على شكل احرف اي ان تحول من int الى char فانك تفعل العكس اي اننا نجمع '0' .

```
int b=9;
char bb=b+'0';
```

جرب تطبيق البرنامج التالي:

```
#include <iostream.h>
void main()
{
	int a =9;
	char m ='6';
	int kk=a+(m-'0');
```

•

```
    char dd=a+'0';
    cout<<kk<<endl<<dd<<endl;
}
```

*** هذه الطريقة تستطيع استخدامها عندما يكون الرقم يتكون من منزلة واحدة فقط اي اننا لا نستطيع تحويل الرقم 130 الى char بهذه الطريقة لانه اكثر من منزلة واحدة والعكس نفس الشيء اي انه لا تستطيع تحويل رقم يتكون من اكثر من منزلة من char الى int بهذه الطريقة.

****

(المكدس)----

(stack)----

المكدس هو عبارة عن مصفوفة(array) ولكن طريقة التحكم بهذه المصفوف يتم بشكل اخر وهي تستخدم مبدأ الذي يدخل اولا يخرج اخرا والذي يدخل اخرا يخرج اولا .انظر المثال الاتي:

عملية ادخال الرقم 5 (cin the number 5) ←

5

عملية ادخال الرقم6 (cin the number 6) ←

6
5

عملية ادخال الرقم7 (cin the number 7)

7 ←
6
5

•

عملية ادخال الرقم8 (cin the number 8)

8
7
6
5

وهنا عملية الادخال تسمى الدفع (push) اي دفع الرقم الى المصفوفة وكما راينا اول شي ادخل الرقم 5 فذهب الى اول عنصر في المصفوفة وهو العنصر 0 وبعد ذلك ادخلنا الرقم 6 ودخل في العنصر الذي بعده وهو العنصر رقم واحد وهكذا .

والان سوف نقوم باخراج هذه البيانات لنر كيف :

اخراج الرقم 8 وشطبه من المصفوفة(cout the number 8 and delete it)

7
6
5

*****ونرى هنا بانه اخر عنصر قمنا بادخاله هو اول عنصر قمنا باخراجه وهذه هي نظرية ال (LIFI) وهي انه الذي يدخل بالاخر يخرج بالاول(last in first out).**

اخراج الرقم 7 وشطبه من المصفوفة(cout the number 7 and delete it)

6
5

اخراج الرقم 6 وشطبه من المصفوفة(cout the number 6 and delete it)

5

اخراج الرقم 5 وشطبه من المصفوفة(cout the number 5 and delete it)

•

*** وكما نرى هنا ايضا ان اول رقم قمنا بادخاله هو اخر رقم قمنا باخراجه.
وهكذا تصبح المصفوفة فارغة.
والان لنفكر ونبدا بتحليل كيف سوف يكون الكود لهذه الفكرة او نظرية المكدس اول شيء سوف نعمل كلاس (class) ولكن ما هي الاشياء التي سوف تكون بهذا الكلاس (class) اول شيء نحتاج الى مصفوفة وسوف نسميها stk و لمتغيرلنسميه count كي يكون عداد كي نحدد عن طريقه رقم العنصر الذي سوف ندخل فيه المعلومات الاتية اذن فسوف تكون قيمته الابتدائية 0 لاننا اول مكان سوف ندخل فيه البيانات هو العنصر 0 وسوف يكونان private وغير ذلك نحتاج الى (counstructer) وبه سوف نعطي القيمة الابتدائية للمتغير count ب 0 .
وبعد ذلك سوف نحتاج الى دالة (function) من اجل ان نقوم بعملية ادخال البيانات (عملية الدفع) وسوف نسميه push ولكن ما هي خصائص هذه الداله اول شيء ماذا سوف تكون قيمتها العائدة هنا يوجد عندنا اكثر من حالتين سوف تكون فيها المصفوفة الحالة الاولى ان المصفوفة مليئة فلن نكون قادرين على ادخال بينات عليها(overflow) والثانية انها سوف تكون فارغة لايوجد بها شيء فلن نستطيع اخذ قيم منها (underflow)والحالة الاخيرة هي انه يوجد بها ولكنها ليست مليئة (success) اذن سوف نستعمل ال enum ولكي نقوم بعملية ادخال البيانات فيجب ان تكون تاخذ قيم(parametares) وهنا سوف نضعها int .
وسوف نحتاج لدالة من اجل الاخراج وسوف نسميها pop وقيمتا العائدة ايضا enum والقيم التي سوف تاخذها ايضا int ولكن المتغير سوف يكون من نوع (refrence) من اجل ان يرجع القيمة التي سوف ياخذها.
وايضا سوف يكون هناك داله من اجل التشييك على المصفوفة هل هي فارغة ام لا وهي سوف تكون قيمتها العائدة من نوع bool وسوف نسميها isempty .

•

والان لنر الكود:

```
#ifndef SATCK_H
#define STACK_H
enum error{succes,un derflow,overflow};
typedef int entry;
const int max=10;
class stack
{
public:
stack();
error push(entry item);
error pop(entry &item);
bool isempty();
private:
int stk[max];
int count;
};
stack::stack()
{
count =0;
}
error stack::push(entry item)
{
error outcome =succes;
if (count>=10)
outcome=overflow;
else
{
stk[count]=item;
count++;
```

•

```
}
return outcome;
}
error stack::pop(entry &item)
{
error outcome=succes;
if(count==0)
outcome=underflow;
else
{
item=stk[count-1];
count--;
}
return outcome;
}
bool stack::isempty()
{
if (count==0)
return true;
else
return false;
}
#endif
```

** هذا هو التعريف وسوف نستخدمه مع جميع برامج ال stack وطبعا هو header file ويجب تضمينه عند استخدام ال stack.

*** او يمكنك من غير عمل header file بان تكتب تعريف ال stack وتكمل في نفس البرنامج كتابة الكود .

نبدأ بالشرح:

•

اول شيء يوجد عندنا ال enum وهنا قد سميناه error وياخذ الثلاث حالات التي نحتاج اليها .
وبعد ذلك عرفنا entry بانها int .
ومن ثم قمنا بتعريف متغير اسمه max وهو من نوعconst int وذلك من اجل ان يكون عدد عناصر المصفوفة عليه وقد وضعناها هنا عشرة .
ومن ثم بدانا بعمل الكلاس (class) ووضعنا اعضاءه (members) .
الان لنبدأ بشرح اعضاء الكلاس(the class members):
اول شيء ال constructer فقط وضعنا فيه القيمة الابتدائية لل count واعطيناه القيمة 0 .

ونأتي الان ل *** push :
اول شيء عرفنا متغير اسمه outcome من نوع error واعطينها قيمة افتراضية (او ابتدائية) success وبعد ذلك بدأنا بجملة if لترى اذا كان ال count اكبر او يساوي عشر فانها سوف اعطي ال outcome قيمة overflow اما غير ذلك فانها سوف تذهب الى جملة ال else وتضع في المصفوفة stk عند العنصر رقم count ال item و ال item هنا هي القيمة التي بعثناها من اجل الادخال ثم تقوم بزيادة ال count واحد .
ومن ثم نرجع قيمة ال outcome .
**اذا بعثنا اول مرة رقم الى ال push فانها سوف تخزنه في العنصر 0 وتزيد قيمة ال count واحد اذن عند عملية الادخال التالية سوف تقوم بالتخزين في العنصر رقم واحد وتزيد ال count واحد ليصبح اثنان وهكذا ....

***pop :
اول شيء عرفنا متغير اسمه outcome واعطيناه قيمة ابتدائية success والان ندخل جملة if وشرطها انه اذا ال count تساوي صفر(فان المصفوفة سوف تكون فارغة) فاذن هنا سوف

•

تعطي outcome قيمة underflow اما غير ذلك فسوف تذهب لجملة else وسوف يضع قيمة العنصر رقم count في المتغير item وبعد ذلك ينقص من قيمة ال count واحد من اجل ان يصبح ياشر على القيمة التالية فعندما نقوم بنفس العملية مرة اخرى فسوف يعطينا القيمة التي تليها وهكذا.

***isempty :

وهي تبدا بجملة If فاذا كان ال count يساوي 0 اذن المصفوفة فارغة فسوف يرجع لنا قيمة true اما اذاكانت غير ذلك فسوف يرجع لنا قيمة false بانها ليست فارغة.

مثال بسيط على ال stack :

```
#include "stack.h"
#include <iostream.h>
void main()
{
stack s1;
int p;
s1.push(5);
s1.push(6);
s1.pop(p);
cout<<p<<endl;
s1.pop(p);
cout<<p<<endl;
}
```

طبعا اول شيء قمنا بتضمين تعريف ال header file الذي لل stack و في ال main بدأنا بتعريف object من نوع stack واسميناه s1 ثم قمنا بعملية push للرقم 5 وبعد ذلك قمنا ايضا بعملية push اخرى للرقم 6 فاصبحت ال stack هكذا:-

6
5

•

وبعد ذلك قمنا بعملية pop ورجعنا القيمة 6 مع المتغير p
فاصبحت ال stack هكذا:-

5

ثم طبعنا قيمة P وهي طبعا 6.
ومن ثم قمنا بعملية pop اخرى ورجع قيمة ال 5 مع ال p
فاصبحت ال stack هكذا:-

اصبحت فارغة فاذا قمت بعملية pop اخرى فانه لن يفعل شيء
لانه سوف يرجع لنا underflow .
-*-وسوف نكتب نفس المثال ولكن من غير ان نعمل ال satck
على ال header file وذلك لمن عنده اشكالية بتلك الطريقة انظر
كيف يصبح:-

```
#include <iostream.h>
enum error{succes,underflow,overflow};
typedef int entry;
const int max=10;
class stack
{
public:
stack();
error push(entry item);
error pop(entry &item);
bool isempty();
private:
int stk[max];
int count;
};
stack::stack()
```

.

```
{
count =0;
}
error stack::push(entry item)
{
error outcome =succes;
if (count>=10)
outcome=overflow;
else
{
stk[count]=item;
count++;
}
return outcome;
}
error stack::pop(entry &item)
{
error outcome=succes;
if(count==0)
outcome=underflow;
else
{
item=stk[count-1];
count--;
}
return outcome;
}
bool stack::isempty()
{
if (count==0)
```

•

```
return true;
else
return false;
}
void main()
{
stack s1;
int p;
s1.push(5);
s1.push(6);
s1.pop(p);
cout<<p<<endl;
s1.pop(p);
cout<<p<<endl;
}
```

كما راينا قمنا بتعريف ال stack بشكل عادي ثم قمنا بكتابة بقية الكود بشكل طبيعي .

_ لناخذ مثال اخر وهنا في هذا المثال سوف نجعل المستخدم يدخل عدد معين هو يضعه من الارقام ومن ثم سوف نضع هذه الارقام في stack وبعد ذلك نطبعها على الشاشة ...

***طبعا لن اكتب تعريف ال stack ولكن يجب عليك انت كتابته.

```
void main()
{
stack s1;
int num , data ,top;
cout<<"**** how much number you want to
enter ****"<<endl;
cin>>num;
cout<<"start enter your numbers :::..."<<endl;
```

•

```
for(int i=0;i<num;i++)
{
cin>>data;
s1.push(data);
}
while (!s1.isempty())
{
s1.pop(top);
cout<<top<<endl;
}
cout<<"*********** ****      <<<<>>>>
**********"<<endl;
}
```

اول شيء قمنا بتعريف object من نوع stack ثم عرفنا المتغير num وذلك من اجل ان نضع فيه عدد الارقام التي يريد ادخالها المستخدم والمتغير data وهو من اجل ان نضع فيه الارقام التي سيدخلها المستخدم وايضا المتغير top وهو من اجل ان ناخذ الارقام من ال stack .

وهناك جملة for وهي من اجل ان نجعل المستخدم يدخل الارقام وفي نفس الوقت ننخزن هذا الرقم في stack وطبعا بدأت جملة for من ال 0 وتنتهي عندما يصبح عدد دوراتها يساوي او اكبر من قيمة ال num (ال num كما قلنا سابقا هي عدد الارقام التي يريد المستخدم ادخالها).

اما جملة while ان شرطها ((! s1.isempty) وكما نعرف ان اشارة ! معناها not فاذا كانت ال stack ليست فارغة فان دالة s1.isempty() سوف ترجع لنا false ومع ال not التي قبلها سوف تصبح true وبما انها اصبحت true فسوف يدخل وينفذ الاوامر التي داخل جملة while وتيقى جملة while في الدوران حتى تصبح ال stack فارغة عندئذ سوف يرجع لنا

•

s1.isempty() قيمة true ومع ال not التي قبلها تصبح false فسوف يخرج من جملة ال while .
وداخل جملة ال while قمنا بعملية pop وطبعنا القيمة التي رجعت منه.
** كما لاحظنا عند تطبيق هذا البرنامج فانه عند عملية اخراج الارقام خرجت بالعكس.
** اذا اردت ان تخرج الارقام بنفس الترتيب الذي ادخلته فبعد ان تدخل الارقام في ال stack انقلهم الى stack اخرى ومن ثم اخرج الارقم من ال stack الاخيرة.
_*_مثال : وهنا سوف نقوم بكتابة برنامج يقوم بتغيير مكان كل عنصريين انظر الجدول للتوضيح:
ال stack قبل

4
5
6
7

ال stack بعد

5
4
7
6

....
الكود:::.....:::

```
void main()
{
        int number , num1;
        stack stack1,stack2,stack3 ;
        cout<<"enter positive numbers"<<endl;
        cin>>number;
        for (int a=0;a<number;a++)
        {
              cin>>num1;
              stack1.push(num1);
```

```
}
while (!stack1.isempty())
{
      stack1.pop(num1);
      stack2.push(num1);

      if(!stack1.isempty())
      {
            stack1.pop(num1);
         stack3.push(num1);
      }

}
while (!stack2.isempty() || !stack3.isempty())
{

if(a%2==0){stack2.pop(num1);stack1.push(
num1);a++;}
      if (!stack3.isempty())
      {
            stack3.pop(num1);
         stack1.push(num1);
      }
      if(!stack2.isempty())
      {
            stack2.pop(num1);
         stack1.push(num1);
      }

}
```

•

```
while (!stack1.isempty())
{
      stack1.pop(num1);
      stack2.push(num1);
}
cout<<"*this is after swaping….."<<endl;
while (!stack2.isempty())
{
      stack2.pop(num1);
      cout<<num1<<endl;
}


}
```

لن اشرح الكود كله ولكن الاشياء المهمة .
جملة ال while الاولى وهنا يتم عملية توزيع ال stack على two stacks وهناك داخل جملة ال while جملة if وذلك من اجل التأكد مرة اخرى اذا كانت ال stack1 قد اصبحت فارغة ام لا.
وجملة ال while الثانية شرطها هو انه اذا كانت احدا ال two stacks ليست فارغة فسوف يدخل على الجملة وفي داخلها لننظر الى اول جملة if انها من اجل اذا كان عدد الاقام المدخلة زوجي فانها سوف تقوم بسحب رقم من stack2 ووضعها في stack1 وزيادة المتغير a واحد وذلك من اجل ان تصبح القيمة فردية فلا ينفذها الا مرة واحدة اذا كان عدد الارقام المدخلة زوجي حاول ان تشطب جملة ال if هذه وتتبع البرنامج ولتر ما سيحصل.
**عندما اقول تتبع البرنامج اعني امسك ورقة وقلم وارسم ما يحصل في ال stacks .

•

وجملة ال while الثالثة من اجل قلب ال stack واعادتها الى الوضع الاصلي ايضا سوف ترى ما يحصل عند تتبعك للبرنامج.
وجملة ال while الاخيرة من اجل طباعة ال stack .
**وهكذا قد اتمينا المكدس (stack) وملاحظة اخيرة اذا اردت ادخال حروف الى المكدس فيجب عليك ان تغير نوع ال array من int الى char وان تجعل ال entry من نوع char ايضا.

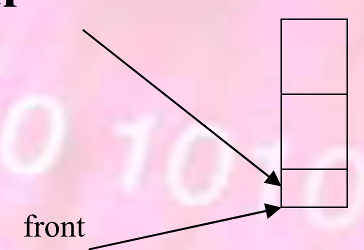
(queue)---

(الطابور)---

وهنا هي عكس المكدس وتستخدم نظرية الذي يدخل اولا يخرج اولا(fifo) first in first out .

لنر الشكل التوضيحي::..::

rear



ونرى هنا بان ال frontو ال rear في البداية ياشران على نفس العنصر وهو العنصر الاول.

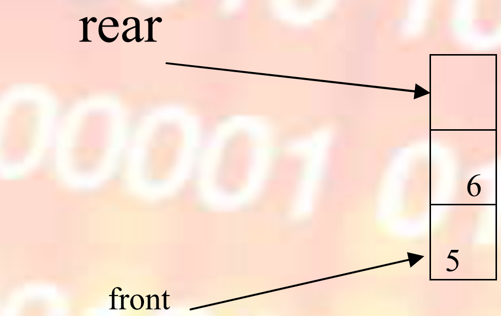
rear

5

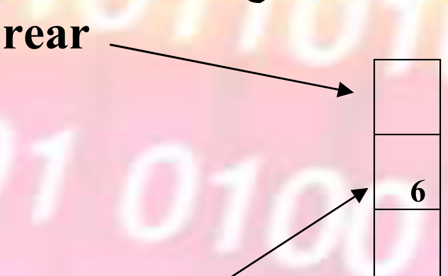
front

وهنا نرى انه عند اضافة عنصر بان ال front يبقى في مكانه ولكن ال rear يصبح ياشر على العنصر الذي بعده اذن فاننا نضيف العناصر عن طريق ال rear . وعند اضافة رقم اخر.

•



وعند اخراج البيانات فانه يخرج من ال front :



هنا قمنا باخراج الرقم 5 فاصبح ال front ياشر على العنصر الذي يليه.

وهناك ثلاث انواع من ال queue :

النوع الاول وهو ال linear :

وهو كما راينا انه يتم اضافة العناصر عن طريق ال rear وسحبها عن طريق ال front ولكن هذه الطريقة تاخذ عدد ارقام معين وهو على حجم المصفوفة فقط .

النوع الثاني وهو ال physical :

وهذا النوع يقوم بنفس الشيء ولكن هنا ال front لا يتغيرر مكانها وتبقى ثابتة طوال الوقت تاشر على العنصر 0 فعندما نريد حذف عنصر نحذفه ولكن بقية العناصر كلها تتحرك الى الاسفل اي انه يصبح العنصر رقم واحد في 0 والعنصر رقم 2 في واحد وهكذا طبعا هذه الطريقة معقدة وغير مفيدة وتأخذ وقت طويل عند التنفيذ.

النوع الثالث وهو ال circular :

وهذا النوع نفس مبدأ ال linear ولكنه عندما تمتلئ المصفوفة فان ال rear يرجع ويأشر على العنصر 0 وهكذا باستطاعتنا ان ندخل العدد الذي نريده من الارقام ولكن طبعا عندما يعيد من ال 0 ونبدا بادخال البيانات فانه سوف يتم وضع القيم مكان القيم التي كانت موجودة

وهذا النوع هو احسن نوع من ال queue على شكل مصفوفات.

•

وفي ال queue نحتاج الى دالة من اجل ان نقوم بعملية ادخال البيانات وسوف نسميها append واخرى من اجل اعادة القيم وسوف نسميها retrieve وايضا الدالة التي تعمل على فحص المصفوفة اذا كانت فارغة ام لا وسوف نسميها isempty وفي ال private سوف نعرف ال count و rear و front و المصفوفة واسمها qu .

وهذا هو تعريف ال queue :

```
enum error {success,underflow,overflow};
const int max=10;
typedef int entry;
class queue
{
public:
        queue();
        error append(entry item);
        error retrieve(entry &item);
        bool isempty();
private:
        int rear,count;
        int front;
        entry qu[max];

};
queue::queue()
{
        count=0;
        front=0;
        rear=max-1;
}
error queue::append(entry item)
{
        error outcome=success;
        if (rear==max-1)
                rear=0;
```

•

```
        qu[rear]=item;
        rear++;
        return outcome;
}
error queue::retrieve(entry &item)
{
        error outcome=success;
        if (rear==front)
                outcome=underflow;
        else
        {
                item=qu[front];
                front++;
        }
        return outcome;
}
bool queue::isempty()
{
        if (rear==front)
         return true;
    else
         return false;
}
```

سوف نبدأ بشرح الدوال :

constructer :

هنا قمنا باعطاء القيم الاولية للمتغيرات ولكن انظر ال rear لم نجعلها تساوي الصفر بل جعلناها تساوي ال max-1 وسوف تعرف السبب عند شرح ال append .

append :

في جملة ال if الشرط هو اذا كن ال rear يساوي ال max-1 فهنا بان ال rear تاشر على اخر عنصر بالمصفوفة اذن نريدها ان تبدا من الاول اي ان نجعلها تاشر على العنصر 0 فقمنا بمساواتها بالصفر اذا تحقق الشرط.

•

** اذن المصفوفة لن تتعبئ ابدا فكلما يصل ال rear الى اخر عنصر فانه عند اضافة عنصر جديد سوف يبدا بالادخال من اول المصفوفة.
وبعد ذلك قمنا بوضع قيمة ال item (وهي القيمة التي بعثناها له) في عنصر المصفوفة عند ال rear ثم قمنا بزيادة ال rear بواحد فعندما نريد ادخال رقم جديد فيدخله بالعنصر الذي بعده وهكذا.

retrieve :

وهنا يوجد عندنا في جملة ال if الشرط بانه اذا كان ال rear وال front متساويين فبهذه الحاله سوف تكون المصفوفة فارغة فنرجع قيمة underflow اما غير ذلك فانه بال else يعطي قيمة ال item قيمة المصفوفة عند ال front وبعد ذلك يقوم بزيالدة ال front واحد كي عندما ناتي المرة الاخرى يكون ال front ياشر على القيمة التي تليه.

isempty :

هنا في جملة الشرط انه اذا كان ال rear و ال front متساويان فان المصفوفة سوف تكون فارغة فيرجع لنا قيمة true وغير ذلك تكون المصفوفة غير فارغة فيرجع لنا قيمة false .

مثال بسيط على ال queue :

المستخدم يدخل ارقام ومن ثم نخزنها في ال queue وبعد ذلك نقوم بطباعة ال queue .

```
#include <iostream.h>
enum error {success,underflow,overflow};
const int max=10;
typedef int entry;
class queue
{
public:
        queue();
        error append(entry item);
        error retrieve(entry &item);
        bool isempty();
private:
        int rear,count;
        int front;
        entry qu[max];
```

```
};
queue::queue()
{
        count=0;
        front=0;
        rear=max-1;
}
error queue::append(entry item)
{
        error outcome=success;
        if (rear==max-1)
                rear=0;
        qu[rear]=item;
        rear++;
        return outcome;
}
error queue::retrieve(entry &item)
{
        error outcome=success;
        if (rear==front)
                outcome=underflow;
        else
        {
                item=qu[front];
                front++;
        }
        return outcome;
}
bool queue::isempty()
{
        if (rear==front)
         return true;
```

•

```
   else
        return false;
}
void main()
{
        queue q1;
        int num,numbers,ret;
        cout<<"enter how much numbers you want to
enter::...:: "<<endl;
        cin>>num;
        cout<<"start enter your numbers::...::"<<endl;
        for(int i=0;i<num;i++)
        {
                cin>>numbers;
                q1.append(numbers);

        }
        cout<<endl;
        while (!q1.isempty())
        {
                q1.retrieve(ret);
                cout<<ret<<endl;
        }
        cout<<"***********   <<>>
************"<<endl;
}
```

في ال for قمنا بادخال الارقام على ال queue وفي ال while قمنا بطباعة ما بداخل ال queue .

وكما نرى بان الناتج الذي طبع لنا على الشاشة قد خرج بنفس الترتيب الذي ادخلناه وذلك بسبب قاعدة ال fifo ان الذي يدخل اولا يخرج اولا حاول تتبع البرنامج ولترى ما يحصل.

•

--وفي هذا المثال سوف نعمل كلاس (class) الطابور العادي ولكن سوف نعمل علاقة وراثة (inhertance) ويكون في الكلاس الثاني دالة لحساب الحجم ودالة اخرى من اجل تفرغ الطابور .
الكود::....:

```
#include<iostream.h>
typedef int queue_entry;
const int maxqueue=10;
enum error_code {success,underflow,overflow};
class queue
{
public:
        queue();
        error_code append(const queue_entry &item);
    error_code retrieveserve(queue_entry &item);
        bool empty()const;
protected:
        int count,front,rear;
        queue_entry entry[maxqueue];
};
queue::queue()
{
        count=0;
        front=0;
        rear=maxqueue-1;
}
bool queue::empty()const
{
        bool outcome=true;
        if(count>0)
                outcome=false;
        return outcome;
}
```

•

```
error_code queue::append(const queue_entry &item)
{
	error_code outcome=success;
	if(count>=maxqueue)
		outcome=overflow;
	else
	{
		count++;
		if((rear+1)==maxqueue)
			rear=0;
		else
			rear++;
		entry[rear]=item;
	}
	return outcome;
}
error_code queue::retrieveserve(queue_entry &item)
{
	error_code outcome=success,outcome2=success;
	if (count<=0)
		outcome=underflow;
	else
	{
		item=entry[front];
		outcome=success;
	}
	if (count<=0)
		outcome2=underflow;
	else
	{
		count--;
		front=((front+1)==maxqueue)?0:(front+1);
	}
```

```
        return (outcome==outcome2)?outcome:underflow;
}
class extended_queue:public queue
{
public:
        int size()const;
        void clear(extended_queue &qq);
};

int extended_queue::size()const
{
        return count;
}
void extended_queue::clear(extended_queue &qq)
{
        while (!qq.empty())
        {
                int a;
                qq.retrieveserve(a);
        }

}
void main()
{
        extended_queue qqq;
        queue_entry dd;
        for (int i=0; i<5;i++)
        {
        cin>>dd;
        qqq.append(dd);
        }
        cout<<" its size ::..:: "<<qqq.size()<<endl<<endl;
```

•

```
        cout<<((qqq.empty())?" empty queue.....":" queue
its not empty\n")<<endl;
        queue_entry rd;
        qqq.retrieveserve(rd);
        cout<<" front number   "<<rd<<endl<<endl;
        cout<<" size is: "<<qqq.size()<<endl<<endl;
    cout<<" the queue now :"<<endl;
        while (!qqq.empty())
        {
               qqq.retrieveserve(rd);
               cout<<rd<<" ";
        }
        cout<<endl;
        qqq.clear(qqq);
}
```

ملاحظات عن الطابور :

*** لا تستطيع ان تقلب عناصر ال queue مثلما كنا نفعل بال stack لانه بال queue دائما الذي يدخل اولا يخرج اولا فلا نستطيع ان نقلبها ب queue اذا اردنا قلبها فيجب ان نستخدم مصفوفة او stack .

***لا يوجد امثلة كثيرة على ال queue ولكنني اظن ان المثال السابق كفى لشرحها جيدا.

****
****
****
****
****

•

(طرق الحسابات الرياضية)---

يوجد لدينا ثلاث طرق لحساب العمليات الرياضية :

الطريقة الاولى (infix): وهي الطريقة العادية التي نستخدمها .

ex:

4+5-6=3.

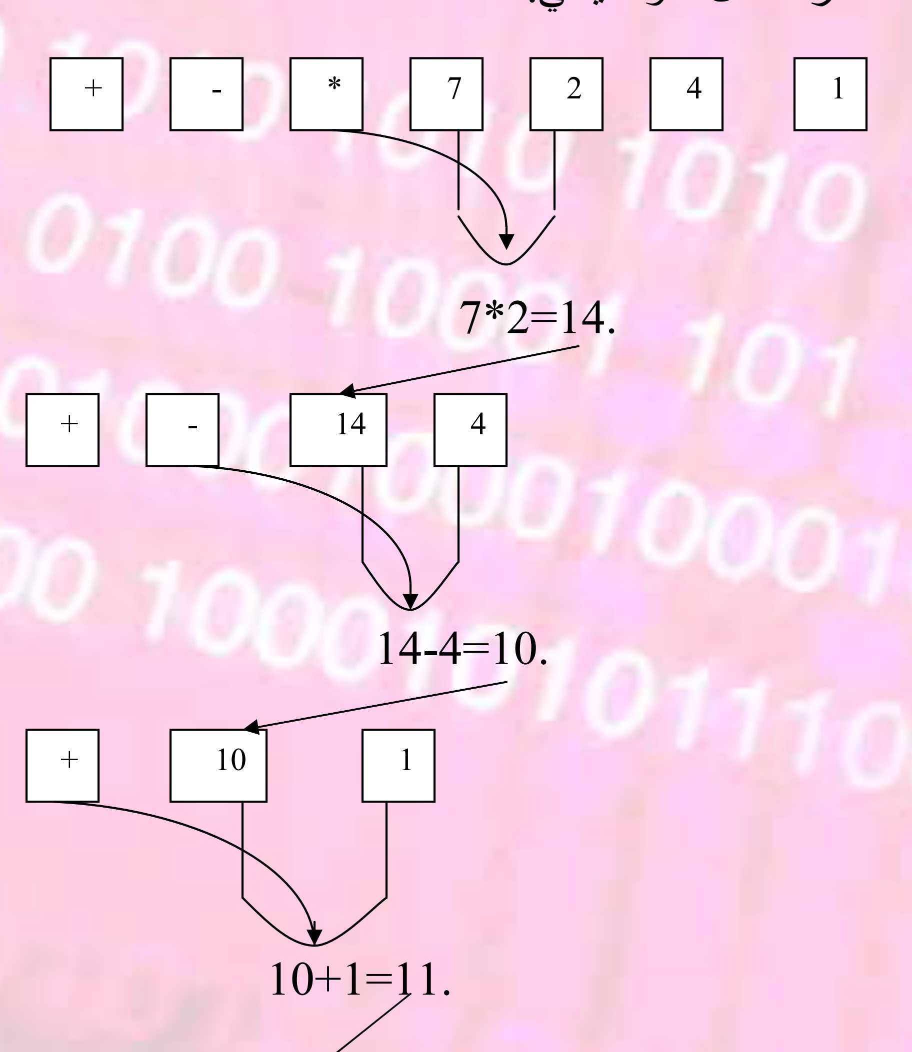
اذن هي الطرقة التي نعرفها جميعا ونستخدمها دائما.

الطريقة الثانية (prefix) : وهذه الطريقة تعزل الاشارات (اي اشارات العمليات) عن الارقام وتكون الاشارات على الجهة اليمنى وتكون الارقام على الجهة اليسرى .

ex:

+ -*7241=11.

ما الذي حصل هنا لنر الشكل التوضيحي:



•

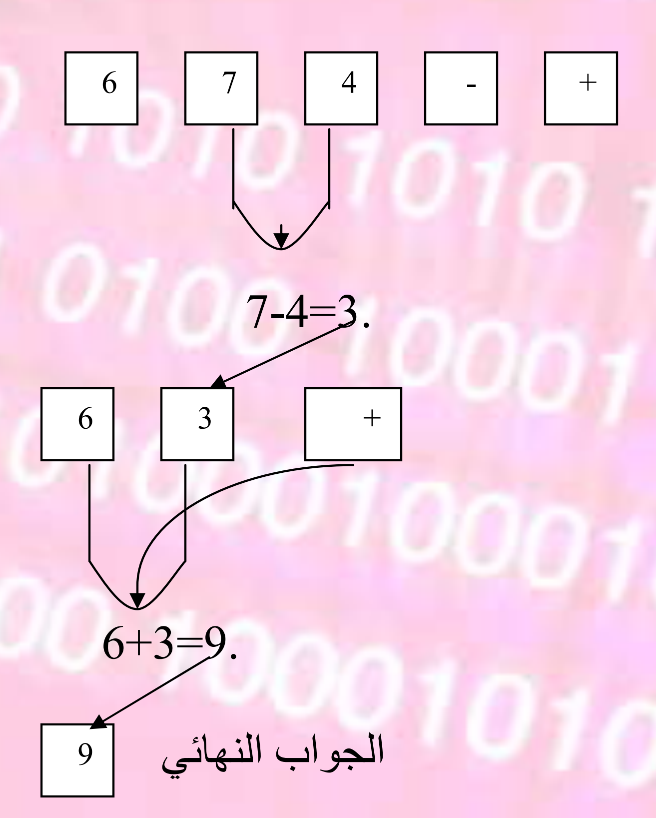
*** ونلاحظ:اننا عندما نقوم بالعملية فاننا ناخذ الرقم الذي على الجهة اليمين ونعمل العملية عليه وطبعا اذا لم نفعل ذلك فان النتيجة ممكن ان تكون مختلفة لان عملية الطرح تؤثر على الناتج في ذلك فبالمثال السابق لو اننا عند 4-14 بدلناها لاختلفت النتيجة .

الطريقة الثالثة (postfix) : وهذه الطريقة عكس السابقة فالارقام توضع على الجهة اليمنى والاشارات على الجهة اليسرى.

ex:

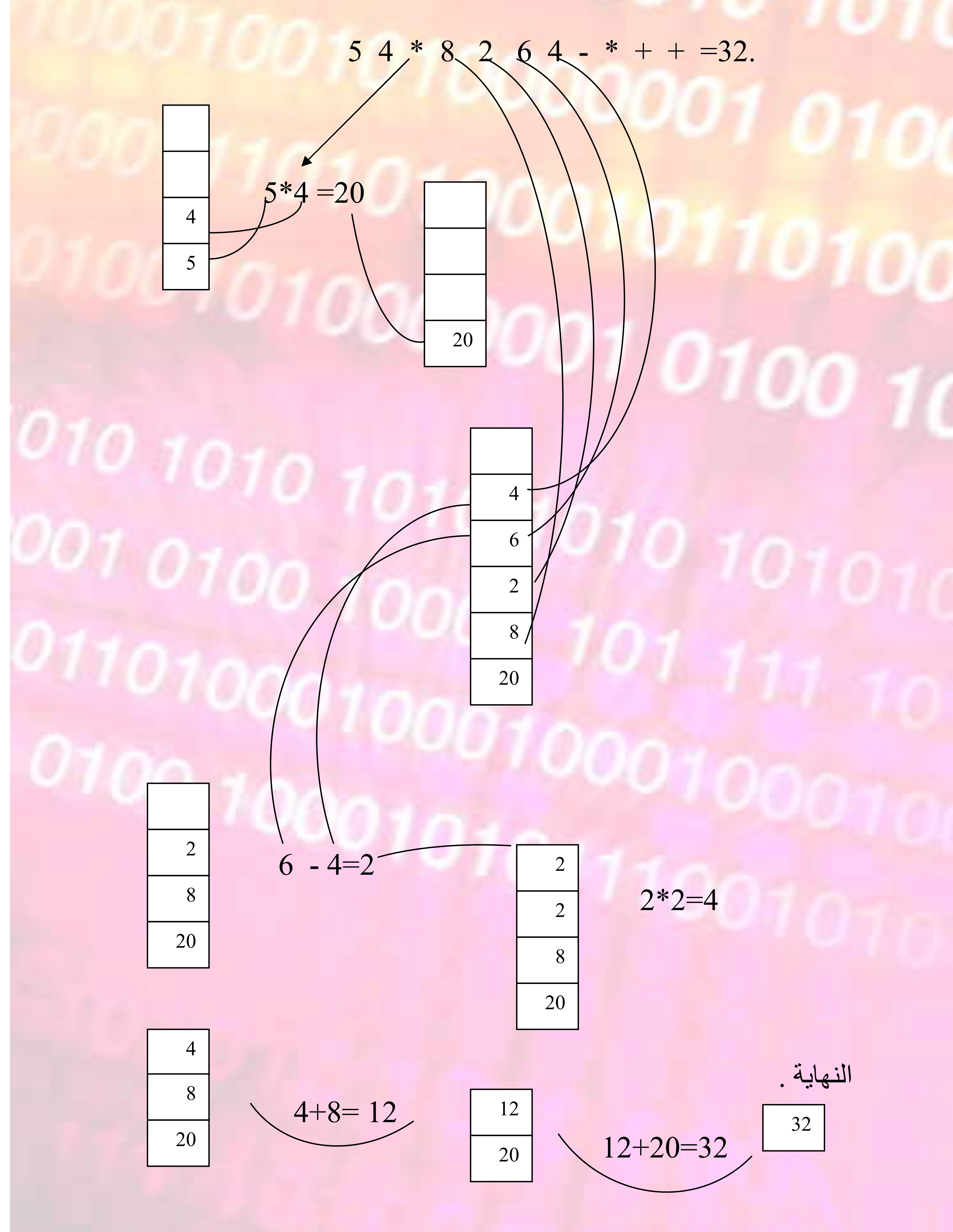
674-+=9.

لنر الشكل التوضيحي لكيفية ايجاد النتيجة:



والان سوف نرى كيف نوجد القيمة عن طريق استخدام المكدس (stack) :

•

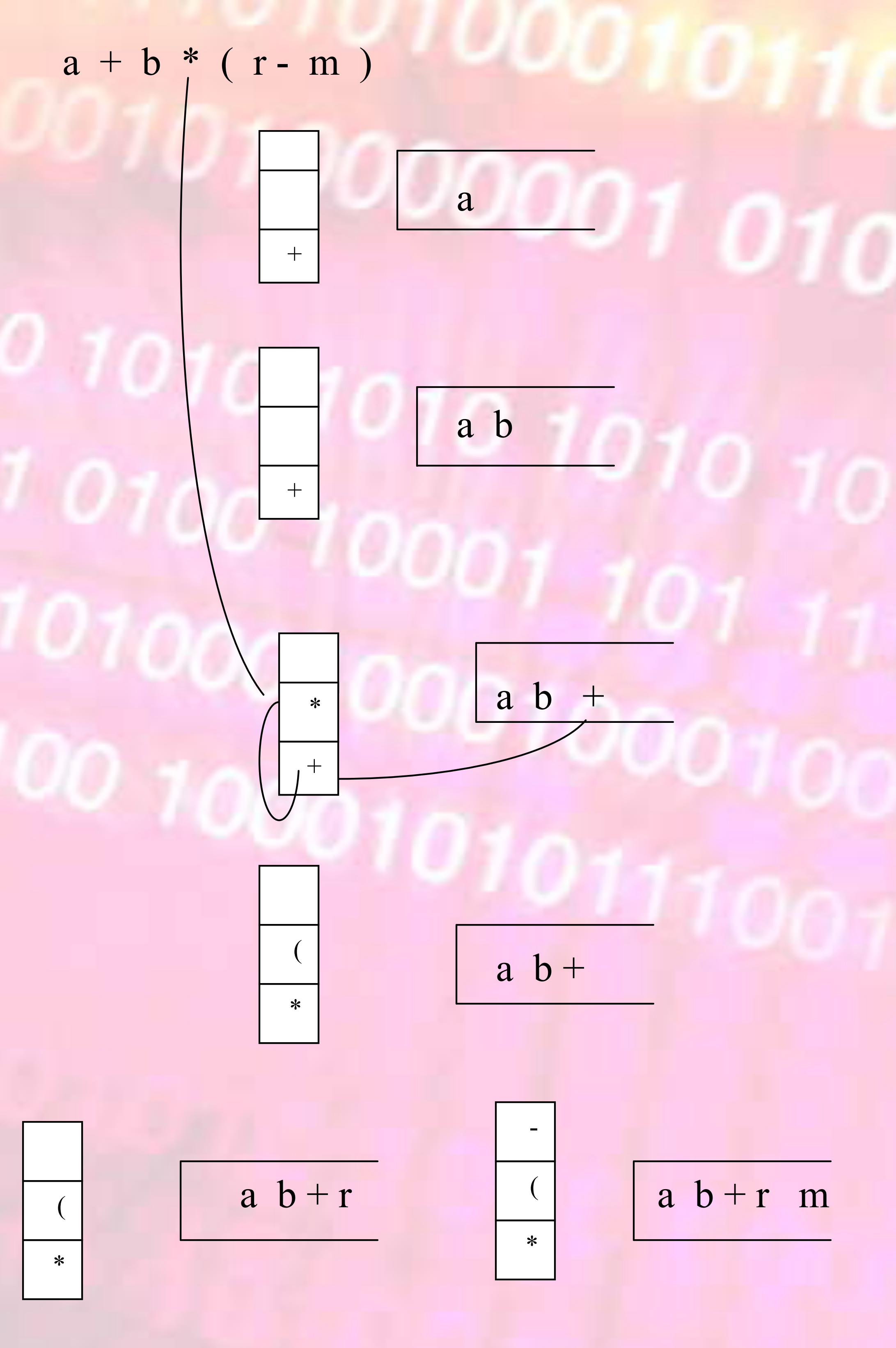


5 4 * 8 2 6 4 - * + + =32.
4
5
5*4 =20
20
4
6
2
8
20
2
8
20
6 - 4=2
2
2
8
20
2*2=4
4
8
20
4+8= 12
12
20
12+20=32
النهاية .
32

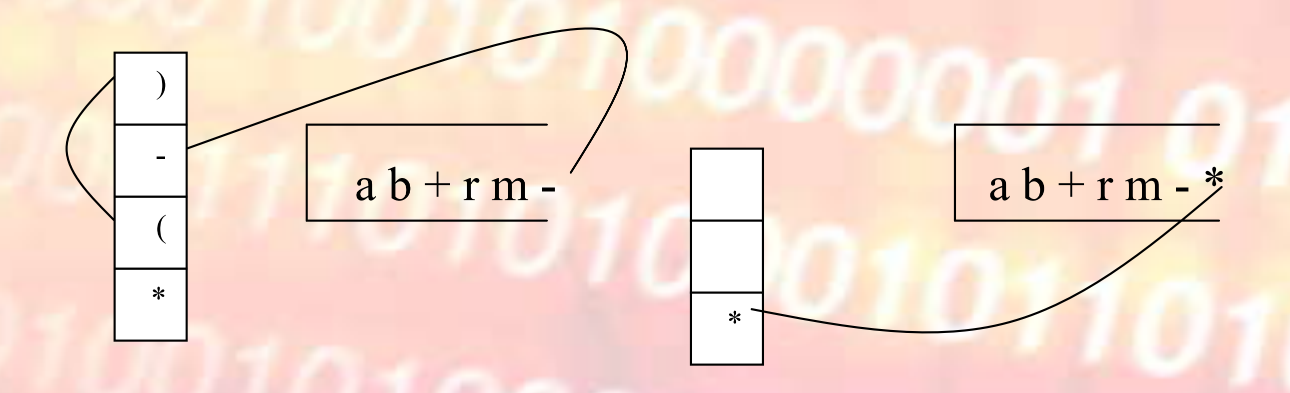
•

وكما راينا في الشكل السابق انه اول شيء ندخل الارقام على المكدس وعندما تاتي اشارة نطبق العملية على اول عنصران في المكدس والناتج نعيد وضعه في المكدس .

وهنا سوف نرى كيف نحول من معادلة infix الى معادلة postfix .



•



اول شيء نضع المتغيرات او الارقام على الجانب وكلما تاتي اشارة نضعها في المكدس.
ولكن هناك قواعد وشروط عند دخول المتغيرات وخروجها وهي:

1- اذا اتت اشارة اولويتها اكبر او تساوي التي اسفل منها فان الاشارة السفلى تخرج عند المتغيرات وطبعا تبقى الاشارة التي اولويتها اكبر بالنزول حتى تصبح اولويتها اصغر من التي اسفل منها او لا يصبح تحتها شيء (وتستثنى الاقواس من هذه القاعده).
2- عند دخول قوس الى المكدس فانه يبقى موجود حتى ياتي القوس الذي يسكره فاذا اتى فانه يخرج الاشارات التي بينهما على الترتيب (من اعلى الى اسفل) فاذا كان هناك اكثر من قوس متداخلات فكل قوس يذهب مع التسكيره التي له.
3- عندما لا يبقى متغيرات ولا اشارات نخرج الاشارات التي داخل المكدس بالترتيب من اعلى الى اسفل.

** عند ادخال الاشارات نسال انفسنا قبل ان ندخل الاشارة هل الاشارة التي باعلى المكدس اويتها اصغر ام اكبر من التي نريد ان ندخلها ام تساويها فاذا كانت اصغر او تساويها فان الاشارة التي بالمكدس تخرج الى عند المتغيرات وتبقى هذه الحال حتى يصبح المتغير الذي نريد ان ندخله اولويته اصغر من الذي موجود في اعلى المكدس .

•

(المؤشرات)---
(pointers)---

هنا مراجعة سريعة للمؤشرات لاننا سوف نقوم باستخدامها في القوائم المترابطة (linked list) بكثرة .

الفرق بين ال pointer وال reference :

ال reference يرجع لك قيمة ال address فهو ليس موقع لمتغير فانت باستخدامه من اجل ان تاخذ ال address لموقع متغير معين وهكذا.

اما ال pointers فهي عبارة عن مؤشر ياشر على موقع معين من الذاكرة ونستطيع التحكم بهذا الموقع عن طريق هذا الؤشر بان ناخذ قيمته او نغيرها.

لنر امثلة كي نوضحهم :

```
#include <iostream.h>
void main()
{
    int a;
    int *pon;
    a=5;
    cout<<&a<<endl<<a<<endl;
    pon=&a;
    *pon=10;
    cout<<&a<<endl<<a<<endl;
}
```

عرفنا المتغير a ومن ثم قمنا بتعريف المتغير pon من نوع مؤشر pointer وضعنا القيمة 5 في المتغير a والان عند ال cout اول شيء يطبع لنا قيمة address وذلك لاننا طلبنا منه ان يرجع لنا قية ال adreess a في جملة &a ومن ثم طبع لنا قيمة ال a .

وفي جملة pon=&a انه قلنا له اجعل المؤشر pon يؤشر على ال address تبع ال a فاصبح المؤشر pon يؤشلر على الa .

وفي جملة pon=10* اننا قلنا له هل ترى الموقع الذي ياشر عليه المؤشر pon اجعل قيمته تساوي عشرة.

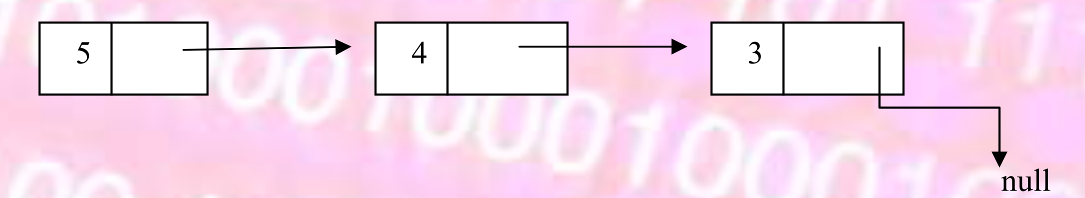
وفي جملة ال cout الاخيرة عندما يطبع لنا نفس ال address ولكن هنا قيمة الa تغيرت واصبحت تساوي عشرة وذلك لاننا غيرنها عن طريق

•

المؤشر فكما نرى اننا نستطيع ان نتحكم بالبيانات التي بالذاكرة عن طريق المؤشات ونغيرها ولكننا لا نستطيع ذلك مع ال refernce فهو فقط يعطينا الموقع في الذاكرة.
**وهذا يكفي بالنسبة للمؤشرات فهذا ما يلزمنا من اجل القوائم المترابطة(linked list) .

---(القوائم المترابطة)
---(linked list)

كانت تواجهنا مشكلة في المصفوفات ان عددها محدود ولا نستطيع التغيير من عدد عناصرها في اي وقت فاذا وضعنا لها عدد معين فلا نستطيع زيادته او التعديل عليه اثناء البرنامج فكان الحل هو القوائم المترابطة وهي ممكن ان نقول عناه بانها علبة فيها شيئان الشيء الاول هو متغير الذي نضع فيه البيانات والشيء الثاني هو عبارة عن address وهو يؤشر على العلبة الذي تليه فهو عبارة عن علب مترابطة عن طريق ال address فبامكاننا بهذه الطريقة ان نخزن فيها عدد لا متناهي من البيانات من غير ان نضع عدد معين لها.
وللتوضيح هكذا سوف يكون مظهرها:



اذن فاول شيء يوجد عندنا العلبة التي فيها الرقم 5 وايضا ال address الذي لها يؤشر على الذي يليها وهكذا واخر علبة التي فيها الرقم 3 ال address الذي لها يؤشر على null اي فراغ او قيمة فارغة.
تعريف كود القوائم المترابطة:

```
#include <iostream.h>
typedef int entry;
struct node
{
	entry data;
	node *next;
```

•

```
        node();
        node(entry _data,node *link=NULL);
};
node::node()
{
        next=NULL;
}
node::node(entry _data,node *link)
{
        data=_data;
        next=link;
}
```

شرح الكود:
عرفنا entry من نوع int .
ثم بدأنا ب struct اسمه node وداخل هذا ال struct يوجد لدينا المتغير data وهو الذي نضع فيه البيانات وايضا object من نوع node وهو مؤشر واسمه next وهناك اثنان constructer .
لنلقي نظرة على تعريف ال object *next هذا ال object يكون عندنا عند تعريف كل object اذن في كل object يوجد فيه object اخر
ولكن لماذا ؟
لاننا نحن نريد ان نجعل كل object يؤشر على object بعده ويكون ال next له (ال next هي ال address تبع ال object الذي يليه) يؤشر على ال object الذي يليه.
ويوجد عندنا هنا overloading لل constructer في ال counstructer العادي وضعنا في الnext تساوي null وذلك لانه عندما نريد ان نعرف object ولا نريد ان نضع فيه شيء فلن يكون مربوط بشيء فهنا ال next تكون تؤشر على null اي فراغ.
اما ال contructer الاخر فهو ياخذ متغيران الاول الذي نبعث له القيمة التي نريد تخزينها والثانية هي ال link وهي address ل node معينة وهي التي تريد ربطها بها. ونلاحظ انه في هذا ال constructer وضعنا ال link تساوي null وذلك من اجل اذا اردت ان تعمل node فيها قيم ولكنه غير مربوط بعدها شيء فيكون ال next لها يساوي null .

•

لنأخذ مثال عليه:

```
#include <iostream.h>
typedef int entry;
struct node
{
entry data;
node *next;
node();
node(entry _data,node *link=NULL);
};
node::node()
{
next=NULL;
}
node::node(entry _data,node *link)
{
data=_data;
next=link;
}
void main ()
{
node *list=NULL;
list =new node(5);
node *f;
node *l;
f=new node (4,list);
l=new node (3,f);
cout<<l->next<<endl;
cout<<l->data<<endl;
cout<<l->next->data<<endl;
cout<<l->next->next<<endl;
cout<<l->next->next->data<<endl;
```

•

}

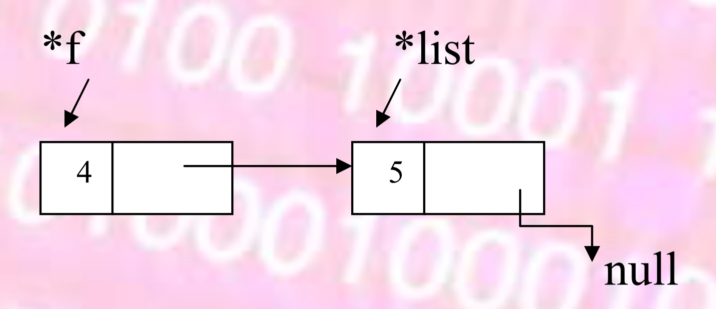
شرح الكود:

اول شيء قمنا بتعريف مؤشر من نوع node اسمه list زجعلنا هذا المؤشر يؤشر على قيمة null (اي لا شيء) ثم قمنا بتعرف ال object لهذا المؤشر وبعثنا له القيمة 5 .

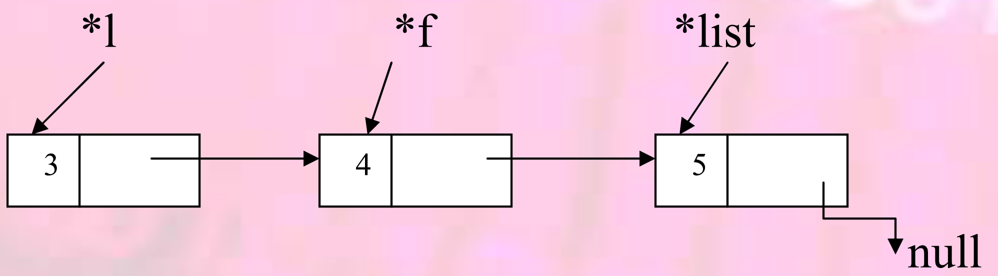
وبما انا بعثنا قيمة فسوف يذهب الى ال overloading constructer وبما اننا بعثنا له القيمة 5 فسوف يضع في الdata قيمة ال 5 وبما اننا لم نبعث له القيمة الثانية(والتي هي address ل node اخرى) فانه سوف يعطيه قيمة null كما نرى في تعريفه .

ثم عرفنا مؤشران اخران من نوع node اسماهما f وL .

وبعد ذلك بدأنا بتعريف ال object للمؤشر f وبعثنا له القيمة 4 وlist فهنا سوف ياخذ القيمة 4 ويضعها في ال data ويأخذ قيمة الlist وهي address ويضعها في ال next فهكذا تصبح هذه ال node يوجد فيها القيمة 4 وتؤشر على node ال list والتي فيها القيمة 5 وتؤشر على null انظر الرسمة للتوضيح.



وبعد ذلك قمنا بتعرف object للمؤشر L زبعثنا له القيمة 3 و ال F فاصبح هكذا عندنا node جديدة فيها القيمة 3 وتأشر على node ال f و node ال f تاشر على ال list انظر الشكل للتوضيح اكثر.



وبعد ذلك يوجد عندنا جملة cout ولكن انظر ما فيها

(cout<<l->next<<endl;)وهنا لنر ما الذي سيحصل:

•

اول شيئء يدخل جملة الcout ثم يرى object ال L وعندنا هنا في هذه الجملة(l->next) نقول له اي انه ارجع لنا قيمة ال next التي داخل node ال L وكما نعرف ان ال next يكون فيها address للnode التي بعدها وهنا سوف يرجع لنا ال address ل node ال F .
**ملاحظة : اشارة ال <- هي عبارة عن اداة مساعده كي نستطيع الدخول الى محتويات ال object لأننا هنا نتعامل مع object لstruct وهذا ال object من نوع مؤشر.
وبعد ذلك يوجد عندنا جملة cout لل (l->data) فهنا سوف يرجع لنا قيمة ال data التي في node ال L اذن سوف يرجع لنا قيمة 3 .
وفي الجملة التي بعدها هي جملة cout فيها (l->next->data) وهنا سوف يبدأ من ال L ويدخل الى ال next التي لل L وال next لل L هنا هي address ال node التي بعدها وهي ال F ومن ثم قلنا له اخرج لنا هذه ال data التي في هذا ال address .اي كأننا نقول له هل تر ال L اذهب لل node التي بعدها واعطيني ال data التي بها. اذن هنا سوف يرجع لي ال data التي بال f اذن سوف يرجع لي القيمة 4.
وبعد ذلك جملة cout بها (l->next->next) فهنا ياتي لل L ويرى قيمة ال next في node ال L والتي هي addressال f ويذهب اليها فيصبح عند node ال f ولكننا لم ننتهي فيوجد هناك next اخرى فيذهب الى قيمة ال next التي بال f وهي قيمة ال address ل node ال list اذن فسوف يرجع لنا ال address لل node ال llist .
والسطر الذي بعده يوجد نفس ما بالسطر السابق ولكن زيادة عليه يوجد عندنا (->data) فهنا عندما يصل الى لل node ال list يدخل الى ال data التي بها ويرجعها لنا اذن هنا سوف يرجع لنا قيمة 5.

مثال اخر::..::** طبعا ضع تعرف ال node قبل ال void main .

```
void main ()
{
node *list=NULL;
int n,r;
cout<<"how much numbers you want to
enter::..::"<<endl;
```

•

```
cin>>n;
for (int i=0;i<n;i++)
{
cin>>r;
node *now=new node(r,list);
list=now;
}
while(list!=NULL)

{
cout<<list->data<<endl;
list=list->next;

}

}
```

شرح الكود:::::
قمنا بتعريف مؤشر من نوع node وهو يؤشلر على null ومن ثم جعلنا المستخدم يدخل عدد الارقام التي يريد ادخالها ووضعنا القيمة في المتغير n ومن ثم يوجد جملة for وهذه الجملة تدور على عدد ال n .
وداخل الجملة for نبدأ بجعل المستخدم يدخل الرقم ومن ثم قمنا بتعريف node اسمه now وقمنا بعمل object له وبعثنا له قيمة ال r و address ال list فهكذا اصبح عندنا node اسمها now فيها data ال r وال next لها به address ال list .
والان بم اننا نريد ان نوصل حميع ال node ببعضها فهنا يوجد عندنا انه دائما يجعل لنا ال next هي ال list اذن فيجب علينا ان نضع ال list في الاول اي ان يؤشر على الاول فهنا وضعنا جملة list=now وبما اننا نتعامل مع مؤشرات فان ال listسوف يصبح ال addressلها مثل ال now وبذلك اصبح ال list يؤشر على اول node فعندما ال for تعيد الدورة تصل ال now الجديدة بال list وهكذا فتصبح عندنا كلها مترابطة.
الان نريد طباعة هذه ال node ولكن كيف استطيع علم عدد ال nodes في هذه القائمة يوجد طريقين:

•

1- ان تاخذ قيمة ال n وهي العدد الذي ادخله المستخدم وجعلناه يدخل فيها عدد القيم التي يريد ادخالها وهذه الطريقة ضعيفة نوعا ما لانه ممكن ان يحصل تغيير على قيمة هذا المتغير.
2- انه يوجد عندنا بأخر القائمة قيمة null وهي موجودة فقط في مكان واحد وهي باخر القائمة ويوجد عندنا المؤشر list يؤشر على اول القائمة فهنا نعمل جملة while يكون شرطها ما دام (list!=null) وداخل الجملة while اول شيء نقوم بطبع ال data عن طريق الجملة ;cout<<list->data وبعد ذلك نريد ان نجعل ال list تؤشر على ال node التي بعدها وذلك عن طريق الجملة ;list=list->next هنا الذي عملناه باننا قلنا له اجعل قيمة ال address لل list هي قيمة ال address الذي في ال next له وكما نعلم ان قيمة ال address في ال next له هي قيمة address ال node الذي بعده فهكذا اصبح المؤشر list يؤشر على ال node الذي بعده.

فعندما تلف ال while مرة اخرى تطبع قيمة ال node التي هي فيه ثم تجعل ال list يؤشر على ال node التي بعده وتبقى على هذه الحال حتى تصبح ال list تؤشر على اخر node فتصبح قيمة ال list تساوي null فهنا عندما يريد الدخول الى جملة ال while فانه لا يحقق الشرط فيخرج منها .

- مثال :::: نريد ان ندخل قيم في قائمة عددها يدخله المستخدم ثم يدخل القيم ثم نريد ان نشطب اول node واخر node ثم نطبع القائمة ::::
الكود::.::

```
void main ()
{
      node *dad=NULL;
      int y,x;
      cout<<"number of numbers::..::"<<endl;
      cin>>y;
      for (int i=0;i<y;i++)
      {
           cin>>x;
           node *now=new node(x,dad);
```

•

```
        dad=now;
     }
    node *p;
    p=dad;
    dad=dad->next;
    delete p;
    node *r=dad;
    while(r->next->next!=NULL)
    {
        r=r->next;
     }
    node *m=r->next;
    r->next=NULL;

   delete m;
   while (dad!=NULL)
   {
     cout<<dad->data<<endl;
     dad=dad->next;
   }

}
```

شرح الكود::::

لغاية اخر جملة ال for لا يوجد شيء جديد فهي مثل المثال الفائت فقد قمنا بادخال قيم في القائمة وكما نرى انه بعد نهاية جملة for يوكن المؤشر dad هو الذي يؤشر على اول القائمة.

الان كي نشطب اول node اول شيء نعرف مؤشر جديد واسميناه p ومن ثم نجعل هذا المؤشر يؤشر على نفس المكان الذي يؤشر فيه ال مؤشر dad وفعلنا ذلك من خلال الجملة ;p=dad الان نريد ان نجعل المؤشر dad يؤشر على ال node التي بعدها فحصل هذا من خلال الجملة ;dad=dad->next الان اصبح عندنا dad يؤشر على ال nodeالثانية و المؤشر p يؤشر على اول node الان كل ما علينا فعله لشطب اول node هو ان نشطب المؤشر p وذلك من خلال الجملة ;delete p .

•

الان نريد شطب ال node الاخيرة وكي نشطبها يجب اول شيء ان نوصل مؤشر الى ال Node قبل الاخيرة وايضا مؤشر الى ال node الاخيرة وذلك من اجل ان نجعل ال next لل node قبل الاخيرة يساوي null ومن ثم نشطب المؤشر الذي على ال node الاخيرة وذلك تم عن طريق الخطوات الاتية:

قمنا بتعريف مؤشر اخر اسمه r وجعلنا يساوي dad فاصبح المؤشر r يؤشر على اول القائمة الان نريد ان نوصله الى ال node قبل الاخيرة وذلك فعلناه عن طريق جملة ال while ولكن لاحظ شرط جملة ال while هو (r->next->next!=null) وذلك لان ال node الاخيرة يكون ال nextلها null اذن ال node قبل الاخية سيكون ال null فيها بال next الذي بال next التي لها.اذن ستبقى جملة while تدور حتى يصل المؤشر r الى ال node قبل الاخيرة .

ثم قمنا بتعريف مؤشر اخر اسمه m وهو يؤشر على ال node التي بعد ال node التي يؤشر عليها المؤشر r اذن اصبح المؤشر m يؤشر على اخر node .

وبعد ذلك جعلنا قيمة ال next لل r يساوي null فاصبحت اخر node بالقائمة هي r .

والان نشطب المؤشر m وذلك من خلال جملة ;delete m .

وطباعة القائمة في جملة ال while وهي مشروحة في مثال سابق.

مثال :::: طباعة قائمة ولكن نريد طباعتها بالعكس اي انبدأ بطباعة اخر قيمة ثم بالروجوع الى الوراء::::

الكود::.:::

```
void main ()
{
      node *list=NULL;
      int num,data;
      cout<<"number of numbers::..::"<<endl;
      cin>>num;
      for (int i=0;i<num;i++)
      {
         cin>>data;
         node *wq=new node(data,list);
```

•

```
        list=wq;
    }
    int m=list->data;
    node *p,*t,*l;
    p=list;
    t=list;
    l=list;
    while(t->next!=NULL)
    {
        t=t->next;
    }
    cout<<t->data<<endl;
    while(p->next!=t)
    {
        p=p->next;
    }
    cout<<p->data<<endl;
    t=list;
    while(p->data!=m)
    {
        while(t->next!=p)
        {
            t=t->next;
        }
        cout<<t->data<<endl;
        p=t;
        t=l;
    }

}
```

شرح الكود:

اول شيء سوف اشرح ماذا نريد ان نفعل اول شيء نريد ان نوصل مؤشر الى اخر node ونطبع ما بها وايضا نوصل مؤشر اخر الى ال node قبل

•

الاخيرة ونطبع ما بها ومن ثم نستخدم هاذان المؤشران حتى نرجع الى اول node .
حتى جملة ال for الاولى قمنا بادخال البيانات .
عرفنا مؤشران من نوع node الاول اسمه p والثاني اسمه t وساوينا المؤشران بالمؤشر list فاصبح هكذا عنا الثلاث مؤشرات يؤشرو على اول node في القائمة.
الان نريد ان نوصل المؤشر t الى اخر node في القائمة وذللك حصل في جملة ال while الاولى فكان الشرط (t->next != null) اي مادام المؤشر t ال next له لا يساوي null فيبقى يدور في الجملة وفي كل دورة يقدم المؤشر t يقدمه node الى الامام وذلك عن طريق الجملة (t=t->next) . فهكذا اوصلنا المؤشر t الى اخر node اذن الان نطبع ال data التي داخلها وبعد ذلك نريد ان نوصل المؤشر p الى ال node قبل الاخيرة و ال node قبل الاخيرة الان نميزها بان ال next لها هو adderss ال node التي يأشر عليها المؤشر t اذن الشرط في جملة ال while الثانية هو (p->next !=t) اي مادام المؤشر p ال next له لا يساوي address ال t فانه يدخل جملة الدوران وداخل جملة الدوران نجعل المؤشر p يؤشر على ال node التي بعدها وذلك بالجملة (p=p->next) . فهكذا اصبح المؤشر p يؤشر على ال node قبل الاخيرة اذن نطبع ال data التي بها .
هكذا طبعنا ال data في اخر node و ال node قبل الاخيرة واصبح المؤشر t يؤشر على ال node الاخيرة والمؤشر p على ال node قبل الاخيرة الان بعد ذلك نريد ان نطبع بقية ال data ولكن ايضا بالرجوع للوراء.
الان جملة ال while الاخيرة شرطها (p->data!=list->data) وكما نعرف ان p يؤشر على ال node قبل الاخير و list يؤشر على ال node الاولى اذن الشرط يقول مادام ال data التي بال p لا تساوي ال data التي بال List فادخل الى الجملة في داخل جملة ال while يوجد عندنا جملة while اخرى وهي من اجل ان تجعل المؤشر t يؤشر على ال node التي قبل ال node التي يؤشر عليها p .وبعد ذلك نقول له اطبع ال data التي في ال node التي يؤشر عليها t لان t اصبح يؤشر على ال node التي قبل ال p . ثم نقول له اجعل المؤشر p يؤشر على المكان الذي يؤشر عليه t فهكذا اصبح المؤشر p يؤشر على ال node التي قبل التي كان يؤشر عليها فهكذا ارجعنا المؤشر p ارجعناه node الى

•

الوراء . ثم قلنا له ان اجعل المؤشر t يؤشر على المكان الذي يؤشر عليه المؤشر list فهكذا يصبح t يؤشر على اول القائمة فعندما يرجع الى جملة الدوران مرة اخرى يكون ال p اصبح على ال node التي طبعت ال data التي لها ونوصل ال t الى ال node التي قبله ونطبع ال data التي به وتبقى في الدوران حتى نصل الى خر node فيصبح ال data في ال p وال data في ال list متساويان فلا يحقق الشرط فانه يخرج من الجملة.
فهكذا نكون طبعنا القائمة من الاخر الى الاول .

المكدس (stack) بالقوائم المترابطة :
الكود:
التعريف فقط.

```
#include <iostream.h>
enum error {success,underflow,overflow};
typedef int entry;
struct node
{
entry data;
node *next;
node();
node(entry item,node *link=NULL);
};
node::node()
{
next=NULL;
}
node::node(entry item,node *link)
{
data=item;
next=link;
}
class stack
{
```

•

```
public:
stack();
error push(entry item);
error pop(entry &item);
bool isempty();
~stack();
private:
node *top_n;

};
stack::stack()
{
top_n=NULL;

}
stack::~stack()
{int a;
while(!isempty())
{pop(a);}
}
error stack::push(int item)
{
node *new_top=new node(item,top_n);
if (new_top==NULL)
return overflow;
top_n=new_top;
return success;
}
error stack::pop(entry &item)
{
node *old_top=top_n;
if(top_n==NULL)
return underflow;
```

•

```
item=old_top->data;
top_n=old_top->next;
delete old_top;
return success;
}
bool stack::isempty()
{
if (top_n==NULL)
return true;
else
return false;
}
```

شرح الكود :

في ال private يوجد عندنا المؤشر top_n من نوع node وهو يؤشر على اخر عنصر ادخل في القائمة.

ال counstructer :

هنا وضعنا المتغير top_n يساوي null لانه في البداية شوف يكون يؤشر على لاشيء.

ال push :

عرفنا مؤشر جديد اسمه new_top وبعثنا له القيمة التي اتت والتي هي ال item و في مكان ال address بعثنا ال top_n اي ان ال node الجديدة سوف يكون بها قيمة ال item وتؤشر على ال top_n .
وبعد ذلك جعلنا ال top_n تؤشر على نفس المكان الذي تؤشر عليه ال new_top فهكذا نكون قد جعلنا ال top_n تؤشر على اول القائمة ففي كل عملية push نعرف node ثم نوصلة بال top_n ثم نجعل ال top_n تؤشر على اول node في القائمة.

ال pop :

عرفنا مؤشر old_top وجعلناه يؤشر على ال top_n .
ثم في جملة ال if قلنا له اذا top_n تؤشر على null اذن فانه لايوجد في القائمة اي شيء فقلنا له رجع underflow .

•

فاذا لم يتحقق الشرط اي اذا كان هناك عناصر في القائمة فانه سوف يكمل وقلنا له ان اجعل قيمة ال item تساوي قيمة ال old_top->data وهي اخر قيمة في القائمة .
وبعد ذلك جعلنا المؤشر top_n يوشر على ال node الذي بعدها وشطبنا المؤشر old_top فهكذا نكون قد ارجعنا القيمة ومن ثم شطبنا اول node واصبح المؤشر top_n يؤشر على اول القائمة الجديدة.
(القائمة الجديدة هي القائمة بعد شطب ال node).

ال isempty :
في جملة ال if نقول له هل المؤشر top_n يؤشر على null فاذا كان يؤشر على null فانه سوف يرجع لنا ture اي انها فارغة اما غير ذلك فيذهب الى ال else ويرجع لنا false اي انها ليست فارغة.

ال destructer :
قمنا بعمل جملة while وشرطها isempty()! اي انه مادامت القائمة ليست فارغة فانه يدخل وفي داخل جملة ال while قمنا بعملية pop فانه مادامت القائمة ليست فارغة فانه سوف يشطب node حتى تصبح فارغة فهكذا كل node قمنا بعملها تشطب .
وطبعا البعض سوف يسأل لماذا عرفنا المتغير a ؟
عرفناه لان ال pop تاخذ متغير من نوع refrence فيجب ان يكون هناك متغيرعند استدعائه كي ترجع به القيمة .

الطابور (queue) بالقوائم المترابطة:
الكود:

التعريف فقط.

```
#include <iostream.h>
enum error {success,underflow,overflow};
typedef int entry;
```

•

```
struct node
{
entry data;
node *next;
node();
node(entry item,node *link=NULL);
};
node::node()
{
next=NULL;
}
node::node(entry item,node *link)
{
data=item;
next=link;
}
class queue
{
public:
queue();
error append(entry item);
error retrieve(entry &item);
bool isempty();
~queue();
private:
node *front,*rear;

};
queue::queue()
{
front=NULL;
rear=NULL;
}
```

```
queue::~queue()
{
int p;
while(!isempty())
retrieve(p);
}
error queue::append(int item)
{
node *new_rear=new node(item);
if (new_rear==NULL)
return overflow;
if (rear==NULL)
{rear=new_rear;
front=rear;}else{
rear->next=new_rear;
rear=new_rear;
}
return success;

}
error queue::retrieve(int &item)
{
if(front==NULL)
return underflow;
item=front->data;
node *old_front=front;
front=old_front->next;
if(front==NULL)
rear=NULL;
delete old_front;
return success;
}
bool queue::isempty()
```

•

```
{
if (front==rear)
return true;
else
return false;
}
```

شرح الكود :
في ال private عرفنا المؤشران front و rear .

ال construcer :
قمنا بمساواة ال front بال null وال rear بال null وذلك لانه بالبداية لايوجد بداية ليؤشر عليه ال front ولا نهاية ليؤشر عليه ال rear.

ال append :
اول شيء عرفنا المؤشر new_rear وبعثنا له قيمة ال item ولاحظ اننا لم نبعث له address .
في جملة ال if الاولى قلنا اذا كانت ال rear تساوي null (اذن فهي فارغة) فجعلنا ال rear تساوي new_rear (لانه سوف تكون ال node الوحيدة فان ال rear سوف يكون يؤشر عليها) وبما انها ال node الوحيدة فان ال front سوف يكون يؤشر عليها فساوينا ال front بال new_rear .
وفي ال else فانه سوف يكون هناك عناصر في القائمة وسوف تكون ال rear تؤشر على اخر node فجعلنا ال node التي تؤشر عليها ال rear جعلناها تؤشر على node ال new_rear فهكذا ربطنا ال node الجديدة وجعلناها في اخر القائمة .
وبما انه اصبحت ال new_rear هي اخر node في القائمة فاذن يجب ان نجعل ال rear تؤشر عليه فساوينا ال rear بال new_rear .

ال retrieve :
اول شيء في جملة ال if الاولى قلنا له اذا كانت ال front تؤشر على ال null فانها فارغة فاذن ليس هناك قيم في القائمة فيرجع underflow .

•

فاذا كان بها قيم فانه سوف يكمل فنقول له اجعل ال item يساوي front->data فوضعنا في ال item قيمة اول node .
ثم عرفنا المؤشر old_front وساويناه بال front فاصبح المؤشران يؤشران على اول القائمة وبعد ذلك جعلنا ال front تؤشر على ال node التي بعده .
وجملة ال if التي شرطها (front==null) اي انه بعد ما جعلنا ال front تؤشر على ال node التي بعدها ولم يكن هناك node فانه سوف يؤشر على null فاذن فهي فارغة فجعلنا ال rear تساوي null .
ثم شطبنا المؤشر old_front فهكذا اخذنا اول قيمة من القائمة وشطبنا ال node الاولى وجعلنا ال front يؤشرعلى اول node في القائمة الجديدة.
***(القائمة الجديدة هي القائمة بعد شطب ال node الاولى).

ال isempty :
في جملة ال if قلنا اذا كان ال front يساوي ال rear فانها فارغة فنرجع true واما غير ذلك فانها ليست فارغة فنرجع false .

ال destructer :
نفس مبدا ال destructer في ال stack فاننا قلنا له في جملة ال while انه مادامت القائمة ليست فارغة فادخل الجملة وفي داخل جملة ال while نعمل retrieve ففي كل لفة يشطب node حتى تفرغ القائمة فانه لا يدخل جملة ال while فهكذا شطبنا كل node قمنا بعملها.

*************)****************
في المكدس والطابور الذين عملناهم على طريقة المصفوفة array فاننا لم نضع destructer وذلك لان المصفوفة array تكون عناصرها متتابعة اي كلها بجانب بعضها فان النظام الذي على الكمبيوتر يستطيع التحكم بها وشطبها عند نهاية البرنامج اما في القوائم المترابطة فان كل node تكون في مكان بعيد عن الاخر فان النظام هنا لن يستطيع تجميع هذه ال nodes وشطبها لوحده فهنا وجب علينا ان نشطبها يدويا فشطبنها عن طريق ال destructer وذلك بما ان ال destructer ينفذ عند نهاية البرنامج فاننا وضعنا جملة الwhile كي تبقى تعمل retrieve حتى تفرغ القائمة فهكذا نكون قد شطبنا القائمة يدويا
**************)***************

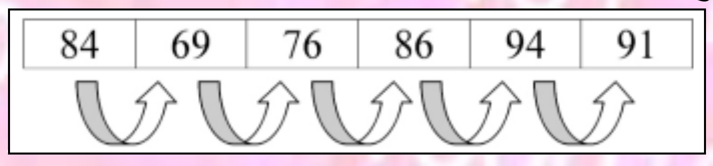
•

********** نهاية القوائم المترابطة **********

(ترتيب البيانات)---
(data sorting)---

ترتيب البيانات وهي هنا المقصود فيها ترتيب الارقام تنازليا او تصاعديا .

اول نوع من الترتيب (bubble sort) :
وهي بمعنى ترتيب الفقاعات وقد اخذت اسمها لانها تترتب تدريجيا او ارتفاعا (اي كالفقاعات) الى اماكنها الصحيحة . اي انها مثل كاس الصودا ترتفع فقاعاته تدريجيا. فهي تقارن كل عنصر بالعنصر الذي بجانبه فيبدأ بالعنصر الاول والثاني واذا كان هناك حاجة للتبديل بمواقعهما فانه يبدلهما ثم عندما ينتهي منهما فانه يكمل بالعنصر الثاني والثالث ويقارن بينهما واذا كان هناك حاجة للتبديل فانه يبدلهما ويبقى على هذه الحاله حتى يصل الى اخر المصوفوفة .



فعندما يصل الى اخر عنصر تعاد العميلة من اول جديد يبدا من اول عنصر والثاني وهكذا ولكن متى هذ العميلة تنتهي ؟
تنتهي عندما ينتهي دورة عملية كاملة ولا يفعل اي تبديل بين العناصر فانه حينها تكون المصفوفة قد رتبت ولهذا استخدمنا المتغير flag للتحكم بجملة الدوران فاذا حصل تبديل لمتغيرين اصبحت قيمته واحد واذا لم يحصل اي تبديل تصبح قيمته صفر.

•

وفي الجدول الاتي هنا مصفوفة يتم ترتيبها بطريقة الفقاعات
-: (bubble sort)

المصفوفة في البداية :-	91	94	86	76	69	84
بعد الدورة الاولى :-	69	91	94	86	76	84
بعد الدورة الثانية :-	69	76	91	94	86	84
بعد الدورة الثالثة :-	69	76	84	91	94	86
بعد الدورة الرابعة :-	69	76	84	86	91	94
بعد الدورة الخامسة:(النهاية)	69	76	84	86	91	94

** ترتيب الفقاعات كتابة الكود له سهل ولكنه ولكنه ابطئ من باقي طرق الترتيب اذ انه يوجد دورة كاملة غير ضرورية لانه اخر مرة يمر ويرى انه لا يوجد اي تبديل يكون قد لف هذه الدورة زياده اي انها غير ضرورية وهذا هو العيب في ترتيب الفقاعات.

الكود:

```
#include <iostream.h>
void bubble_sort();
int a[10];
void main()
{
int num;
int _data;
cout<<"number of numbers ::..::"<<endl;
cin>>num;
for (int i=0;i<num;i++)
{
cin>>_data;
a[i]=_data;
}
bubble_sort();
for ( i=0;i<num;i++)
{
  cout<<a[i]<<endl;
```

•

```
}
}
void bubble_sort()
{
    int i, j, flag = 1;
    int temp;
    int arrayLength = 10;
    for(i = 1; (i <= arrayLength) && flag; i++)
    {
        flag = 0;
        for (j=0; j < (arrayLength -1); j++)
        {
            if (a[j+1] > a[j])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
                flag = 1;
            }
        }
    }
}
```

النوع الثاني ترتيب الادراج:(insertion sort)--

هذا النوع من الترتيب اشبه بترتيب كروت الشدة فانك تجعل منطقة مرتبة ومنطقة غير مرتبة وسحب من المكان الغير المرتب وتضعه في مكانه مرتبا في المنطقة المرتبة .

فهي تقسم المصفوفة الى منطقتان فبالمنطقة الاولى تكون دائما مرتبة والمنطقة الثانية تكون باقي قيم المصفوفة غير المرتبة ونحن ناخذ القيم من جزء المصفوفة غير المرتبة وندخلها الى جزء المصفوفة المرتبة وندخله في مكانه الصحيح اي مرتبا.

جدول توضيح ترتيب المصفوفة من خلال هذه الطريقة:

•

في البداية:	84	69	76	86	94	91
لون المصفوفة الاولى =	84	69	76	86	94	91
لون المصفوفة الثانية=	84	69	76	86	94	91
	84	76	69	86	94	91
	86	84	76	69	94	91
	94	86	84	76	69	91
النهاية	94	91	86	84	76	69

***هذه الطريقة تكون فعالة وسريعة في حالة ان تكون المصفوفة صغيرة الحجم ولكنها تخسر هذه الكفائة عندما تتعامل مع المصفوفة كبيرة.

الكود:

```
#include <iostream.h>
void insertion_sort();
int a[100];
void main()
{
cout<<"number of numbers ::..::  ";
int num;
cin>>num;
int data;
for (int i=0;i<num;i++)
{
cin>>data;
a[i]=data;
}
insertion_sort();
for ( i=0;i<num;i++)
{
cout<<a[i]<<endl;
}

}
void insertion_sort( )
```

•

```
{
    int i, j, key, array_length=100;
    for(j = 1; j < array_length; j++)        {
        key = a[j];
        for(i = j - 1; (i >= 0) && (a[i] < key); i--)
        {
            a[i+1] = a[i];
        }
       a[i+1] = key;
    }

}
```

النوع الثالث ترتيب الاختيار (selection sort) :
هو عبارة عن مزيج من البحث والترتيب فهي تبحث عن القيمة المطلوبة (اكبر قيمة او اصغر قيمة حسبما تريد الترتيب) وتضع هذه القيمة في المكان الصحيح .
عدد الدورات التي يقوم بها هذا النوع من الترتيب هو
(عدد عناصر المصفوفة - 1) .
وفي هذا النوع فانها في اول دورة تاخذ اكبر قيمة(او اصغرها) وتضعه في مكانه وفي الدورة الثانية تاخذ ثاني اكبر قيمة وهكذا...
وهذا الجدول يوضح كيف يتم ترتيب مصفوفة بهذه الطريقة:

المصفوفة في البداية	84	69	76	86	94	91
بعد الدورة الاولى	84	91	76	86	94	69
بعد الدورة الثانية	84	91	94	86	76	69
بعد الدورة الثالثة	86	91	94	84	76	69
بعد الدورة الرابعه	94	91	86	84	76	69
بعد الدورة الخامسة(النهاية)	94	91	86	84	76	69

هذه الطريقة ايضا سهلة كتابة الكود له ولكنها الاقل كفاءة بين كل الطرق اذ انه ليس هناك اي فرصة ان ينتهي الترتيب باكرا فهي يجبب ان تقوم بكل الدورات حتى اذا ادخلت له مصفوفة مرتبة.
الكود:

```
#include <iostream.h>
```

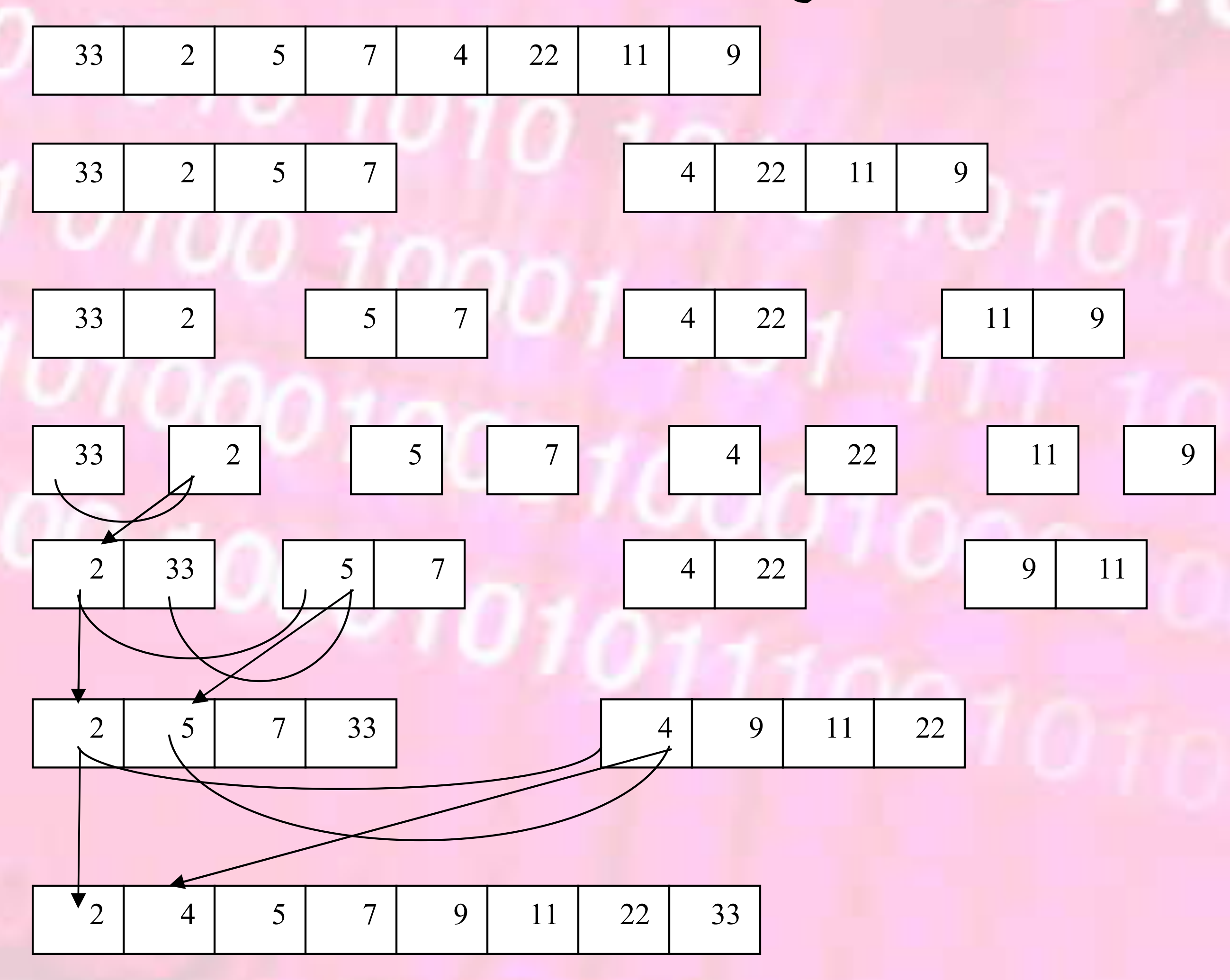
```
void selection_sort();
int a[100];
void main()
{
cout<<"number of numbers ::..::  ";
int num;
cin>>num;
int data;
for (int i=0;i<num;i++)
{
cin>>data;
a[i]=data;
}
selection_sort();
for ( i=0;i<num;i++)
{
cout<<a[i]<<endl;
}
}
void selection_sort()
{
    int i, j, first, temp;
    int array_size = 100;
    for (i= array_size - 1; i > 0; i--)
    {
        first = 0;
        for (j=1; j<=i; j++)
        {
            if (a[j] < a[first])
            first = j;
        }
        temp = a[first];
        a[first] = a[i];
```

•

```
    a[i] = temp;
  }
  return;
}
```

رابع نوع : الدمج (merge sort) :
هذا النوع هو عبارة عن تقسيم المصفوفة تدريجيا حتى يصبح كل عنصر لوحده وعند الانتهاء من التقسيم تصبح عملية دمج العناصر وعند الدمج فاننا ندمج العناصر مرتبة ونبقى بعملية الدمج تدريجيا حتى ترجع المصفوفة كما كانت ولكن مرتبة.

انظر الرسمة للتوضيح اكثر:



•

فكما نرى من الرسمة السابقة انه عندما انتهى من تقسيم المصفوفة وعندما بدأ بالدمج بدأ يقارن العناصر فياخذ الاصغر ويدخلة وهكذا حتى قمنا بدمج كل العناصر .

**وهي من اكثر الطرق فعالية في كل الحالات وهي سريعة التطبيق .

```
#include <iostream.h>
void merge(int start, int last);
void mergeSortStub(int start, int last);
int a[100];
void main()
{
cout<<"number of numbers ::..::  ";
int num;
cin>>num;
int data;
for (int i=0;i<num;i++)
{
cin>>data;
a[i]=data;
}
mergeSortStub(0,num-1);
for ( i=0;i<num;i++)
{
cout<<a[i]<<endl;
}

}
void merge(int start, int last)
{
int i, j, k;
int aux[100];
int mid = (start + last) / 2;
```

•

```
for (i = start; i <= mid; i++)
aux[i] = a[i];
for (i = mid+1; i <= last; i++)
aux[last+mid+1-i] = a[i];
j = start; k = last;
for (i = start; i <= last; i++)
a[i] = (aux[j] < aux[k]) ? aux[j++] : aux[k--];
}

void mergeSortStub(int start, int last)
{
if (last > start) {
int mid = (last + start) / 2;
mergeSortStub(start, mid);
mergeSortStub(mid+1, last);
merge(start, last);
}
}
```

****

---(البحث)

(searching)---

في البحث يوجد عندنا طريقتان:

1-البحث الخطي المتتالي:

وهو يبدا من اول المصفوفة ويبحث في كل عنصر ويقارنه بالمطلوب فاذن فانه يبدا من اول عنصر ويبقى حتى يجده فمن الممكن ان يجده اول عنصر ومن الممكن ان يجده اخر عنصر فاذن اسوأ حالة انه سوف يجده اخر عنصر فانه سوف يلف على عدد العناصر الموجودة وهي طريقة ليست ضعيفة ولكنها بطيئة نوعا ما خاصة اذا اردت ان تبحث في مصفوفة كبيرة.

الكود:

```
#include<iostream.h>
```

```
int a[10];
int sequential_search(int key);
void main ()
{
int number,ret;
cout<<"enter  your   numbers :"<<endl;
for (int i=0 ; i<10;i++)
{
cin>>number;
a[i]=number;
}
cout<<"enter the number you want to search for:
"<<endl;
cin>>number;
ret=sequential_search(number);
if (ret==-1)
cout<<"sorry what you need not here ::"<<endl;
else
cout<<"your number in the element ::  "<<ret<<"
::"<<endl;
}
int sequential_search( int key)
{
      int index = 0;
      while( (index < 10) && (key != a[index]))
      {
            if (a[index] != key)
                  index++;
      }
  if (index==10 )

index=-1;
return (index);
```

•

}

2-البحث الثنائي (النصفي):
وسمي النصفي لانه يبدأ من نصف ويقارن العنصر مع العنصر الذي بنصف المصفوفة فاذا كان اكبر فانه يبدأ البحث في جزء المصفوفة العلوي اما اذا كان اقل فانه يبحث في النصف السفلي .
**(طبعا يجب ان تكون المصفوفة مرتبة تصاعديا (sorting array)).

الكود:

```
#include <iostream.h>
int a[10];
void binarySearch(int lowerbound, int upperbound,int
key);
void main()
{
int number,ret;
cout<<"enter  your   numbers :"<<endl;
for (int i=0 ; i<10;i++)
{
cin>>number;
a[i]=number;
}
cout<<"enter the number you want to search for:
"<<endl;
cin>>number;
binarySearch(0,9,number);
}
void binarySearch( int lowerbound, int upperbound, int
key)
{
	int position;
	int comparisonCount = 1;
```

•

```
	position = ( lowerbound + upperbound) / 2;

	while((a[position] != key) && (lowerbound <=
upperbound))
	{
		comparisonCount++;
		if (a[position] > key)
		{
			upperbound = position - 1;
		}
		else
		{
			lowerbound = position + 1;
		}
		position = (lowerbound + upperbound) / 2;
	}
	if (lowerbound <= upperbound)
	{
		cout<< "The binary search found the number
after " << comparisonCount << " comparisons.\n";
		cout<< "The number was found in array subscript
"<< position<<endl<<endl;
	}
	else
		cout<< "Sorry, the number is not in this array.
The binary search made "<<comparisonCount << "
comparisons.";
	return;
}
```

*** اضف احد طرق الترتيب (sorting) الى الكود وذلك من اجل انه بعد ادخال المصفوفة نرتبها ثم نبحث عن الرقم .

•

****

(الاستدعاء الذاتي)---

(recursion)---

الاستدعاء الذاتي هو عبارة عن استدعاء دالة لنفسها حتى شرط معين فانه يقف من استدعاء نفسه فهي اذن عبارة عن طريقة للدوران مثل ال for ولكنها بطريقة استدعاء الداله لنفسها .
وهنا يكون عندنا داله يكون داخلها اقل شيء شرطان(ليس شرطان اي جملتا شرط بل انه يوجد حالتان) الشرط الاساسي وهو متى يقف عن استدعاء نفسه والشرط الثانوي وهو يقوم باستدعاء نفسه .
وبما انه عبارة عن طريقة دوران فان الاستدعاء الذاتي ممكن كتابته عن طريق جملة for .
***كل استدعاء ذاتي يمكن ان نعمله على جملة for .
ولكن ليس كل جملة for يمكن عملها على طريق الاستدعاء الذاتي.

المثال الاول:
هنا نريد ان نعمل برنامج لحساب المضروب والمضروب العدد هو عيارة عن ضرب العدد في كل الاعداد التي اقل من حتى تصل الى ال 1 .
فمضروب العدد 5 هو 5*4*3*2*1 =120

اول شيء سوف نكتب البرنامج بجملة ال for ثم نكتبه بطريقة الاستدعاء الذاتي .

```
#include <iostream.h>
void main()
{
int num;
```

•

```
cout<<"please enter the number::..::"<<endl;
cin>>num;
int factorial=1;
for ( ; num!=0;num--)
{
factorial=factorial*num;
}
cout<<factorial<<endl;
}
```

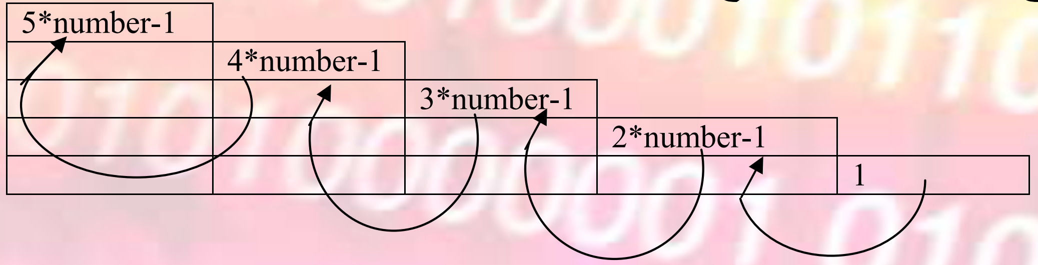
الان سوف نكتبه بجملة الاستدعاء الذاتي :

```
#include <iostream.h>
int factorial( long number)
{
if (number <= 1)
return 1;
else
return number * factorial(number - 1);
}
void main()
{
long num = 0;
 cout<<"Enter in a integer: ";
cin>>num;
cout<<factorial(num)<<endl;
}
```

لنر ما حصل:
بال void main ادخلنا الرقم ثم في جملة ال cout قمنا باستدعاء الدالة factorial وبعثنا له قيم الرقم الذي ادخلناه .
والان في دالة ال factorial يوجد عندنا جملة ال if والشرط فيها انه اذا كانت قيمة ال number (وهي القيمة التي بعثنها) اقل من واحد او اصغر فارجع قيمة واحد وغير ذلك في جملة ال else ارجع قيمة ال number

•

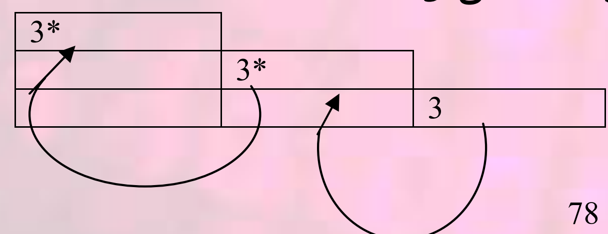
ضرب استدعاء الدالة factorial وبعثنا له قيمة ال number -1 فهكذافاذا بعثنا له قيمة ال 5(وهي قيمة ال number) فانه سيدخل على الدالة فهل قيمة ال number تساوي 1 لا اذن يكمل فيضرب ال 5 باستدعاء ادالة factorial ويبعث لها القيمة اربعة وهكذا حتى يصل ال 1 فانه يرجع لنا قيمة 1 ويتوقف .
توضيح لكيفية سير البرنامج:



فهكذا عندما يصل الى ال 1 فانه يرجع 1 والواحد هاذا يرجع الى ال 2 فتصبح 1*2 والقيمة تصبح اثنين ويرجعها الى 3 فتصبح 2*3 فتصبح القيمة 6 وهذه ال 6 ترجع الى ال 4 فتصبح 6*4 فتصبح القيمة 24 وترجع الى ال 5 فتصبح 24*5 اذن القيمة 120 .
*** ممكن ان نسمي الاستدعاء الذاتي recursion بالعلاقة البنائية المرتجعة لانها تبقى النزول وثم تبدا بالرجوع والبناء الى الاعلى وكل شيء بالاعلى يعتمد على الذي اسفل منه.***

المثال الثاني:
هنا نريد برنامج تعطيه رقم ونعطيه الاس فيعطينا القيمة وذلك عن طريق الاستدعاء الذاتي .
فهنا سوف يكون عندنا انه سوف نعطيه رقم ونعطيه اس وعندما نبعثة لدالة recursivePow فانه سوف يكون عندنا ضرب العدد اول مرة واستدعاء الدالة وبعث الرقم نفسه وننقص من الاس واحد ونبقى في هذه الحالة حتى تصبح قيمة الاس يساوي واحد فهنا اننا نرجع الرقم نفسه (انتبه نرجع الرقم نفسه وليس واحد).

انظر الجدول وقد ادخلنا الرقم 3 وجعلنا الاس 3



•

الكود:-

```
#include <iostream.h>
int recursivePow(int Number, int Exponent);
void main()
{
int Number, Exponent;
cout << "Enter a number.\n";
cin >> Number;
cout << "Enter the power to take the number to.\n";
cin >> Exponent;
cout << Number << " to the power of " << Exponent <<
" is: "<< recursivePow(Number, Exponent) << endl;
}
int recursivePow(int Number, int Exponent)
{
if (Exponent == 1)
return Number;
else
{
Exponent--;
return Number * recursivePow(Number, Exponent);
}
}
```

مثال :

هنا فكرة برج هانوي واساس الفكرة انه يوجد عندنا ثلاث ابراج في البرج الاول يوجد عدد من الاسطوانات وهذه الاسطوانات مرتبة على اساس ان الاسطوانه الكبيرة تكون بالاسفل والاصغر فوقها .

والمطلوب ان ننقل الاسطوانات من البرج الاول الى البرج الثالث ولكن شروط النقل انه عند النقل لا تاتي اسطوانة كبيرة فوق صغيرة اي الحفاظ

•

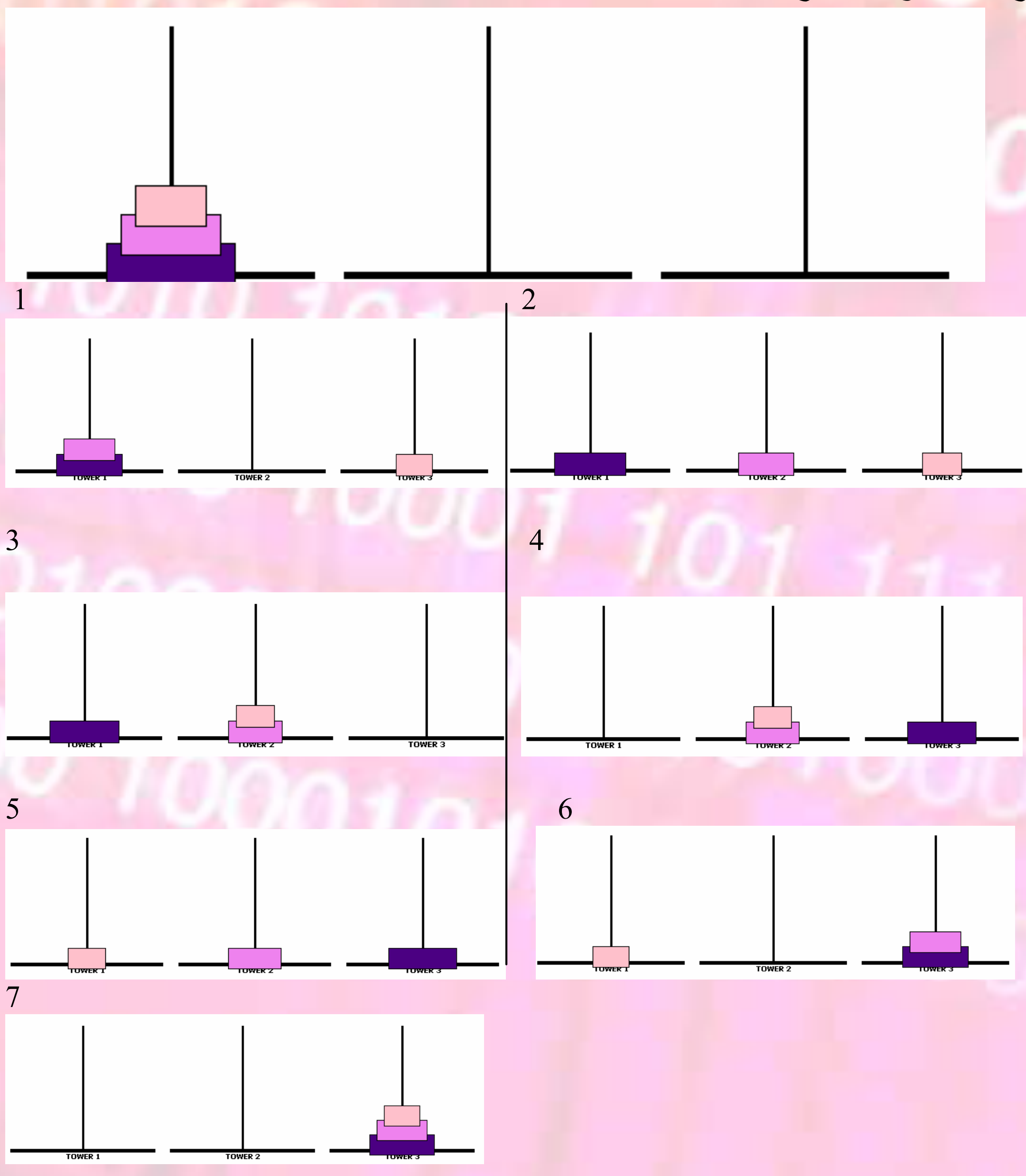
على الترتيب وان ينقلو على البرج الثالث بنفس الترتيب الذي كان في البرج الاول.

وهناك قاعدة لمعرفة اقل عدد من الحركات اذا اردنا وضع عدد اسطوانات معين فاذا كان عدد الاسطوانات n .

اقل عدد من الحركات $=(2^n)-1$

وهذه الرسمة توضح كيف ننقل الاسطوانات وذلك على ثلاث اسطوانات .

ولكي نعرف اقل عدد ةحركات فاننا نعوض بلقاعدة $(2^3)-1$ اذن اقل عدد حركات هو 7 حركات .



الكود :-

•

```
#include <iostream>
#include <cstdlib>
#include <conio.h>
#include <cmath>
using namespace std;
typedef char* Peg;
void move(Peg A, Peg B);
void transfer(size_t N, Peg A, Peg B, Peg C);
void get_disk_num(int &iNum);
int main()
{
while(1)
{
cout << "\t\t\tHanoi Towers Puzzle Solver" << endl <<
endl;
cout << "Enter number of disc on the first peg (enter -1
to quit): ";
int iDiskNum, iStepNum;
get_disk_num(iDiskNum);
if(iDiskNum == -1)
{
cout << "hope you enjoyed using these program!" <<
endl;
break;
}
else
{
iStepNum = pow(2, iDiskNum) - 1;
cout << "the shortest solution can be reach in " <<
iStepNum << " steps" << endl;
cout << "press any key to show the solution...";
getch();
cout <<endl;
```

•

```
transfer(iDiskNum, "Peg1", "Peg2", "Peg3");
system("pause");
system("cls");
}
}
return 0;
}
void move(Peg A, Peg B)
{
cout << "move top most disc from " << A << " to " << B
<< endl;
}
void transfer(size_t N, Peg A, Peg B, Peg C)
{
if(N > 0)
{
transfer(N - 1, A, C, B);

move(A, C);

transfer(N - 1, B, A, C);
}
}

void get_disk_num(int &iNum)
{
cin >> iNum;
if(iNum != -1 && iNum < 1)
{
cout << "please notice that the number of disc needs to
be an integer bigger than 0" << endl;
cout << "number of disc on the first peg: ";
get_disk_num(iNum);
```

•

```
}
}
```

****

كودات متفرقة.

وهذا كود ال (post fix) :

**اريد الاشارة هنا ان تنبته اول شيء الى فصل ال pop الى pop و top فال pop تشطب عنصر فقط وال top نعيد لنا قيمة اول عنصر فقط وثاني شيء لوجود overloading في الداتين top وpush وذلك لانه يوجد لدينا هنا نوعيين من البيانات احداها int وهي الارقام والاخرى char وهي الرموز فقمنا بعمل دالة push كي تدفع القيم التي تكون ارقام وهي تاخذ int ودالة push اخرى كي تدفع الرموز زهي تاخذ cahr وكذلك فعلنا في ال top .

```
#include <iostream.h>
const int maxsize=10;
typedef char stack_entry;
enum Error_code {success,overflow,underflow};
class stack{
public:
stack();
Error_code push(stack_entry &item);
Error_code push(int &item);
Error_code pop();
Error_code Top(stack_entry &item)const;
Error_code Top(int &item)const;
bool empty()const;

private:
int count;
stack_entry x[maxsize];
};
stack::stack()
```

```
{
count=0;
}
Error_code stack::push(stack_entry &item)
{
Error_code outcome=success;
if (count>=maxsize)
outcome=overflow;
else
{
x[count]=item;
count++;
}
return outcome;
}
Error_code stack::push(int &item)
{
Error_code outcome=success;
if (count>=maxsize)
outcome=overflow;
else
{
x[count]=item;
count++;
}
return outcome;
}
Error_code stack::pop()
{
Error_code outcome=success;
if (count==0)
outcome=underflow;
else
```

```
count--;
return outcome;
}
Error_code stack::Top(stack_entry &w)const
{
Error_code outcome=success;
if (count==0)
outcome=underflow;
else
w=x[count-1];
return outcome;
}
Error_code stack::Top(int &w)const
{
Error_code outcome=success;
if (count==0)
outcome=underflow;
else
w=x[count-1];
return outcome;
}
bool stack::empty()const
{
if(count==0)
return true;
else
return false;
}
void main()
{
int w,h,l=0;
char v;
stack s1,s2,s3,s4,s5;
```

•

```
int x,y,z;
cout<<"Enter your input size";
cin>>w;
for (int i=0;i<w;i++)
{
cin>>v;
if (v=='+' || v=='-' || v=='*' || v=='/' )
{
s1.push(v);
}
else
{
s3.push(v);
}
if (v=='+' || v=='-' || v=='*' || v=='/' )
{
switch (v)
{
case '+':
{
s2.Top(x);
s2.pop();
s2.Top(y);
s2.pop();
z=y+x;
s2.push(z);
l++;
break;
}
case '-':
{
s2.Top(x);
s2.pop();
```

•

```
s2.Top(y);
s2.pop();
z=y-x;
s2.push(z);
break;
}
case '*':
{
s2.Top(x);
s2.pop();
s2.Top(y);
s2.pop();
z=y*x;
s2.push(z);
break;
}
case '/':
{
s2.Top(x);
s2.pop();
s2.Top(y);
s2.pop();
z=y/x;
s2.push(z);
break;
}
}//END OF SWITCH
}//END OF IF
else
{
h=(v-'0');
s2.push(h);
}
```

•

```
}//END OF FOR
s2.Top(h);
cout<<(h)<<endl;
}//END OF MAIN
```

--هذا الكود هو عبارة عن تعديل في القوائم المترابطة وهو يجعلنا ان نقوم بانشاء قائمة فارغة ثم ندخل القيم فيها اي اننا اول شيء ننشئ قائمة تحتوي مثلا على 20 node وتكون كل ال nodes فارغة ومتصلة وطبعا هذه الطريقة لا نستطيع عملها بتعريف القوائم المترابطة العادي:

الكود:::...:

والذي حصل هنا اننا غيرنا في ال overloading constructer وجعلناه ياخذ اول شيء ال address وهو يجب ان ياخذ address ولكنا جعلنا القيمة اختيارية فاذا اراد ان يضع قيمة فنسمح له اما اذا لم يرد فاننا نجعلها فارغة اي انها تساوي null .

```
#include <iostream.h>
typedef int entry;
struct node
{
	entry data;
	node *next;
	node();
	node(node *link=NULL,entry _data=NULL);
};
node::node()
{
	next=NULL;
}
node::node(node *link,entry _data)
{
	data=_data;
	next=link;
```

•

```
}
void main()
{
	node *p=NULL;
	int n,r;
	cout<<"how many nodes you want to
make::..::"<<endl;
	cin>>n;
	for(int i=0;i<n;i++)
	{
		node *bb=new node (p);
		p=bb;
	}
	node *f=p;
	while(f!=NULL)
	{
		cin>>r;
		f->data=r;
		f=f->next;
	}
	f=p;
	cout<<endl;
	while(f!=NULL)
	{
		cout<<"   "<<f->data<<endl<<"<<
>>"<<endl;
		f=f->next;
	}

}
```

النهاية بحمد الله.